Introduction to IEEE-754 floating-point arithmetic

Nathalie Revol
INRIA – LIP - ENS de Lyon – France

TMW VII, 14-17 December 2011, Key West

From $\mathbb{R}$ to $\mathbb{F}$

Representing real numbers of a computer using a finite number of bits to approximate them.
For efficiency issues, a finite and fixed number of bits is preferred (32 or 64).

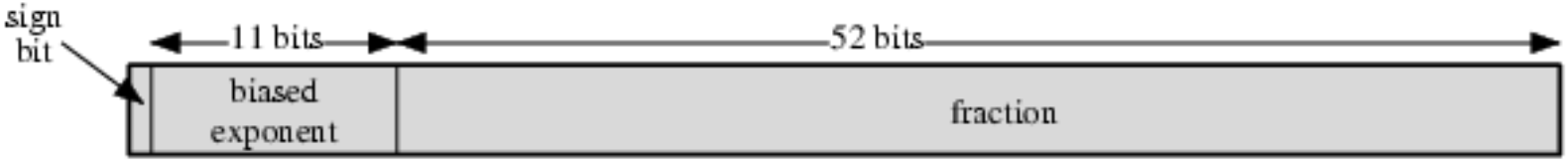
From $\mathbb{R}$ to $\mathbb{F}$

IEEE-754 representation:

- symmetry around 0, being able to represent a number and its opposite: 1 bit for the sign
- dynamics: floating-point representation (ie $m \times 10^e$ or $m \times 2^e$)
- symmetry wrt multiplication, being able to represent a number and its inverse (roughly speaking): the exponent range is symmetric around 0
- hidden bit: the first bit of the mantissa is a "1" (for normal representations) and does not need to be stored
- other considerations (precision)...

IEEE-754 representation for floating-point numbers

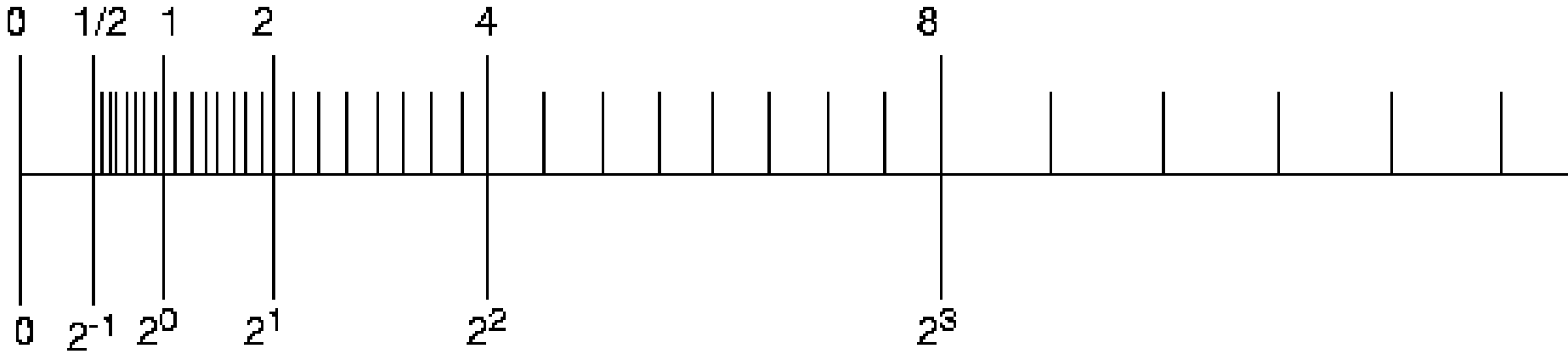
(a) Single format



(b) Double format

IEEE-754: underflow and subnormals

Binary Representation:



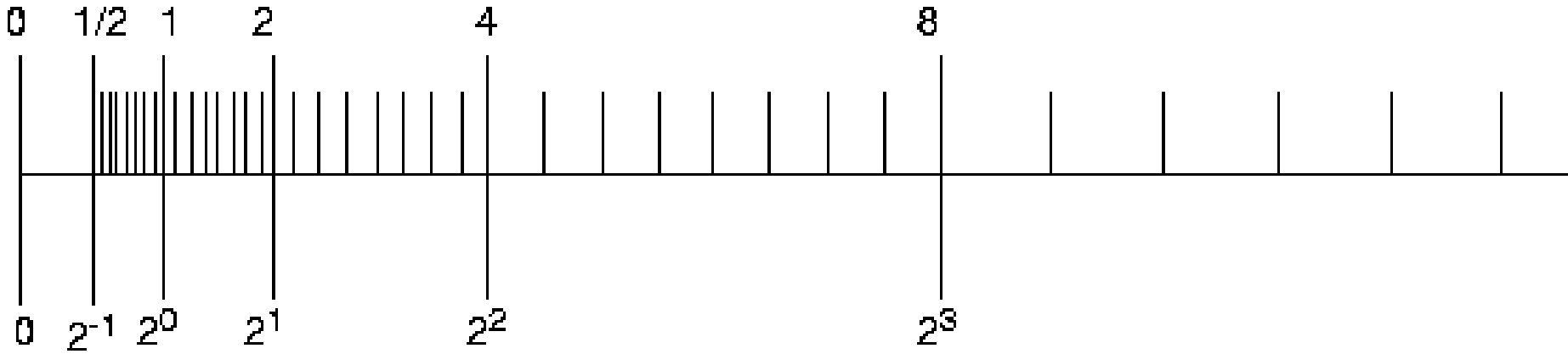
When the exponent is minimal, the rule for the hidden bit does not apply any more: the first bit is explicitely represented. Such numbers are called **subnormals**.

Roundoff error is no more relative but absolute for subnormals.

Specific (and delicate) handling of subnormals in proofs.

IEEE-754: underflow and subnormals

Binary Representation:



When the exponent is minimal, the rule for the hidden bit does not apply any more: the first bit is explicitely represented. Such numbers are called **subnormals**.

Roundoff error is no more relative but absolute for subnormals.

Specific (and delicate) handling of subnormals in proofs.

IEEE-754 representation: special values ∞ and NaN

Remark: 0 has two representations in $\mathbb{F}$: $+0$ and -0.

What happens for MAXFLOAT + MAXFLOAT?
Answer: $+\infty$ and $-\infty$ belong to $\mathbb{F}$.
And thus $1/(+0) = +\infty$ whereas $1/(-0) = -\infty$.

What happens for $0/0$?
Answer: this is an invalid operation, the result is NaN (Not a Number).

IEEE-754 representation: special values ∞ and NaN

Remark: 0 has two representations in $\mathbb{F}$: $+0$ and -0.

What happens for MAXFLOAT + MAXFLOAT?
Answer: $+\infty$ and $-\infty$ belong to $\mathbb{F}$.
And thus $1/(+0) = +\infty$ whereas $1/(-0) = -\infty$.

What happens for $0/0$?
Answer: this is an invalid operation, the result is NaN (Not a Number).

IEEE-754 representation: special values ∞ and NaN

Remark: 0 has two representations in $\mathbb{F}$: $+0$ and -0.

What happens for MAXFLOAT + MAXFLOAT?
Answer: $+\infty$ and $-\infty$ belong to $\mathbb{F}$.
And thus $1/(+0) = +\infty$ whereas $1/(-0) = -\infty$.

What happens for $0/0$?
Answer: this is an invalid operation, the result is NaN (Not a Number).

IEEE-754 representation: special values ∞ and NaN

Remark: 0 has two representations in $\mathbb{F}$: $+0$ and -0.

What happens for MAXFLOAT + MAXFLOAT?
Answer: $+\infty$ and $-\infty$ belong to $\mathbb{F}$.
And thus $1/(+0) = +\infty$ whereas $1/(-0) = -\infty$.

What happens for $0/0$?
Answer: this is an invalid operation, the result is NaN (Not a Number).

IEEE-754: specification of the arithmetic operations

To get

- the best possible accuracy
- reproducible results
- well-specified results
- ...

$a \diamond b$ should return the result as if it were computed exactly and then rounded: this is called *correct rounding*. The operation $\diamond$ can be $+$, $-$, $*$, $/$ or $\sqrt{}$.

Four rounding modes:

- rounding upwards,
- rounding downwards,
- rounding to zero,
- rounding to nearest (tie to even).

IEEE-754: specification of the arithmetic operations

Implementation issue: 3 bits suffice to get correct rounding.

Exact result: if the result is representable using floating-point arithmetic, then this exact result will be returned.

IEEE-754: other issues

Elementary functions (exp, log, sin, atan, cosh, ...): correct rounding is recommended (but not required).

Radix can be 2 or 10.

IEEE-754: properties on the error

With rounding to nearest.

If we compute using FP arithmetic

$$z := \frac{x}{\sqrt{x^2 + y^2}},$$

can we prove that $-1 \leq z \leq 1$?

Can we compute $t = \sqrt{1 - z^2}$ without getting a NaN?

The answer is **yes**.

IEEE-754: properties on the error

With rounding to nearest.

If we compute using FP arithmetic

$$z := \frac{x}{\sqrt{x^2 + y^2}},$$

can we prove that $-1 \leq z \leq 1$?

Can we compute $t = \sqrt{1 - z^2}$ without getting a NaN?

The answer is **yes**.

IEEE-754: properties on the error

Relative error $\leq$ 1 ulp or 1/2 ulp.
ulp (unit in the last place): it corresponds to the weight of the last bit of the mantissa.
In IEEE-754 double precision,

- with rounding to nearest, the relative error of one operation is $\leq 1/2$ ulp,
- with other rounding modes, the relative error of one operation is ≤ 1 ulp,

provided no overflow nor underflow occurs.

IEEE-754: properties on the error

Strictly between x and $x(1 - \mathrm{ulp}(x))$, there can be 0 or 1 floating-point number,
there is 1 if x is a power of 2.

Changes of binades must be handled with care in proofs.

IEEE-754: properties on the error

Strictly between x and $x(1 - \mathrm{ulp}(x))$, there can be 0 or 1 floating-point number,
there is 1 if x is a power of 2.

Changes of binades must be handled with care in proofs.

IEEE-754: properties on the error

Sterbenz lemma:

Let a and b be two positive floating-point numbers. If

$$\frac{a}{2} \leq b \leq 2a$$

then $a - b = a \ominus b$.

In other words, $a - b$ is exactly representable in floating-point arithmetic.

IEEE-754: properties on the error

Theorem:
for any pair (x, y) of FP numbers and for rounding to nearest, there exists FP numbers r_+ and r_- such that

$$\begin{aligned} r_+ &= (x + y) - (x \oplus y) \\ r_- &= (x - y) - (x \ominus y) \end{aligned}$$

Furthermore, r_+ and r_- can be computed using FP operations.

IEEE-754: properties on the error

Why with "rounding to nearest" only?

Counterexample for directed rounding

with basis 2 and at least $p > 4$ bits of mantissa, let's take

$$\begin{aligned} x &= -(2^{2p} + 2^{p+1}) \\ y &= 2^p - 3 \end{aligned}$$

then we have

$$\begin{aligned} x + y &= -2^{2p} - 2^p - 3 \\ x \oplus y &= -2^{2p} \\ (x + y) - (x \oplus y) &= -2^p - 3 = -(2^p + 3) \quad \text{not representable.} \end{aligned}$$

IEEE-754: computing the roundoff error

Let x and y be two normal FP numbers such that $|x| \geq |y|$ and the rounding mode be to nearest. Let also assume that $x \oplus y$ does not overflow.

$$\begin{array}{lrcl} \text{Algo Fast2Sum:} & s & = & x \oplus y \\ & z & = & s \ominus x \\ & r & = & y \ominus z \end{array}$$

The mathematical equality holds:

$$s + r = x + y$$

i.e. r is the roundoff error on the addition of x and y.
Beware of the optimizations done by your compiler...

IEEE-754: computing the roundoff error

The roundoff error can be computed without test (to compare x and y) at the expense of more arithmetic operations.
The roundoff error of a multiplication can also be computed (either using Dekker's algorithm or an FMA).

This means that it is possible to compute using FP arithmetic and twice the computing precision: in **double-double** arithmetic, each number is represented as the sum of two FP numbers (but the FP addition is not performed).

To conclude

IEEE-754 standard for floating-point arithmetic

- specifies the formats and the behaviour of the operations,
- makes it possible to bound roundoff errors, to track them during computations (cf. N. Higham: *Accuracy and Stability of Numerical Algorithms*, SIAM 2002),
- makes it possible to compute the roundoff error,
- makes it possible to compute in higher precision,
- makes it possible to establish proofs on the quality of the results...
  but this is tedious and error-prone (many cases to handle: possibility of overflow, underflow, change of binade...).

Formal Proofs of Polynomial Models

Nathalie Revol
INRIA – LIP - ENS de Lyon – France
Joint work with Pieter Collins and Milad Niqui

14 December 2011

Polynomial models: definition

Notations: $x^{\alpha} = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_v^{\alpha_v}$.
All variables belong to $[-1, 1]$.

Polynomial model: it is a pair $< p, \mathbf{I} >$ where p is a polynomial and $\mathbf{I}$ is an interval.

$< p, \mathbf{I} >$ **represents the function** $f : \mathbb{R}^v \to \mathbb{R}$ if $\forall x \in [-1, 1]^v$, $f(x) - p(x) \in \mathbf{I}$.

Polynomial models: arithmetic

Operations can be defined between polynomial models: addition, subtraction, multiplication by a scalar, multiplication, composition...

They use the corresponding operations on polynomials and on intervals.
(Beware: the degree of the resulting polynomial may vary.)

Polynomial models: sweeping

To get a simpler representation, ie. a polynomial with less coefficients: **sweeping operation**.

The "lost" coefficients are accounted for in the interval remainder: **I** becomes $\mathbf{I} + \sum[-|a_i|, |a_i|]$ where the sum is over swept terms.

Thus manipulated polynomials can keep constant degrees: higher degree terms are swept.

Polynomial models: implementation

To implement polynomial models and arithmetic on a computer: use of floating-point arithmetic to represent the coefficients and to perform arithmetic operations on them.

Floating-point operations are not exact
thus rounding errors have to be accounted for in the interval remainder.
How?

Polynomial models: implementation

Bounding the error of one operation:
if the operation is performed using rounding to nearest, then the error between the exact operation and the rounded-to-nearest operation verifies

$$a \diamond b - RN(a \diamond b) \leq (RU(a \diamond b) - RD(a \diamond b))/2$$

where RN, RU, RD stand for rounding to nearest, upward, downward.

(Again, the division by 2 should be performed using RU.)

Polynomial models: implementation

Example of the addition of two polynomial models

$< p, \mathbf{I} > + < q, \mathbf{J} >$
where $p = \sum_{i=0}^{m} a_i x^{\alpha_i}$ and $q = \sum_{j=0}^{n} b_j x^{\beta_j}$. Then

$$< p, \mathbf{I} > + < q, \mathbf{J} > = < \sum_{k=0}^{\max(m,n)} RN(a_k + b_k) x^{\alpha_k}), \mathbf{I} + \mathbf{J} + \sum_{k=0}^{\max(m,n)} (RU(a_k + b_k) - RD(a_k + b_k))/2 > .$$

Example of lemma used in COSY

Let a and b be two floating-point numbers. Then

$$|(a+b) - RN(a+b)| \leq RN((\mathrm{ulp}(1)/2) * \max(|a|, |b|))$$

provided no overflow nor underflow occurs.

Example of lemma: proof

Case distinction:

- a and b have opposite signs;
- $a = b$;
- a and b are wide apart (assume $|b| < |a|$ and $\mathrm{ulp}(1)/2 \leq \beta \leq 1$);
- $b = (1 - \beta)a$ with $0 < \beta < \mathrm{ulp}(1)/2$ (assume $|b| < |a|$), case distinction again, with 2-3 possible values for b in each case:
  - rounding mode to nearest or not?
  - a is a power of 2 or not?
  - $a-$ (the floating-point immediately closer to 0 than a) is a power or 2 or not?

Example of proof

	rounding to nearest (even) $\varepsilon_m = u$		other rounding modes $\varepsilon_m = 2u$		
	b	error		b	error
$a = 2^t$	a^-	$ua/2 \leq \varepsilon_m \otimes a$		a^-	$ua/2 \leq \varepsilon_m \otimes a$
	a^{--}	$0 \leq \varepsilon_m \otimes a$		a^{--}	$0 \leq \varepsilon_m \otimes a$
				a^{---}	$ua/2 \leq \varepsilon_m \otimes a$
$2^t < a < 2^{t+1}$	a^-	$ua \leq \varepsilon_m \otimes a$	$a^- = 2^t$	a^-	$ua^- \leq \varepsilon_m \otimes a$
				a^{--}	$3ua^-/2 \leq \varepsilon_m \otimes a$
				a^{---}	$0 \leq \varepsilon_m \otimes a$
			$2^t < a^-$	a^-	$u2^t \leq \varepsilon_m \otimes a$
				a^{--}	$0 \leq \varepsilon_m \otimes a$

Do you trust this proof?

Example of proof

	rounding to nearest (even) $\varepsilon_m = u$		other rounding modes $\varepsilon_m = 2u$		
	b	error		b	error
$a = 2^t$	a^-	$ua/2 \leq \varepsilon_m \otimes a$		a^-	$ua/2 \leq \varepsilon_m \otimes a$
	a^{--}	$0 \leq \varepsilon_m \otimes a$		a^{--}	$0 \leq \varepsilon_m \otimes a$
				a^{---}	$ua/2 \leq \varepsilon_m \otimes a$
$2^t < a < 2^{t+1}$	a^-	$ua \leq \varepsilon_m \otimes a$	$a^- = 2^t$	a^-	$ua^- \leq \varepsilon_m \otimes a$
				a^{--}	$3ua^-/2 \leq \varepsilon_m \otimes a$
				a^{---}	$0 \leq \varepsilon_m \otimes a$
			$2^t < a^-$	a^-	$u2^t \leq \varepsilon_m \otimes a$
				a^{--}	$0 \leq \varepsilon_m \otimes a$

Do you trust this proof?

Proof checker

What is it?

It is as if a very pernickety reader checks your proof.

As it is not human, no chance that it forgets one case, nor that it admits an implicit assumption.

But sometimes this can be tedious: you have to explain to it that $2_{\mathbb{N}} = 2_{\mathbb{R}}$ . . . and many such things.

Proof checker

What is it?

It is as if a very pernickety reader checks your proof.

As it is not human, no chance that it forgets one case, nor that it admits an implicit assumption.

But sometimes this can be tedious: you have to explain to it that $2_{\mathbb{N}} = 2_{\mathbb{R}}$... and many such things.

Proof checker

The proof is checked in its deep details until the computer agrees with it.
Often, use of formal proof checkers, meaning programs that only check a proof (they may also generate easy demonstrations).
Therefore the checker is a very short program (de Bruijn criteria : the correctness of the system as a whole depends on the correctness of a very small kernel).
(Courtesy Sylvie Boldo)

Why use a proof checker?

Even if polynomial models already provide validated computations. . .
this provides one extra level of confidence.

Proof checker

Which proof checker? The Coq proof assistant (http://coq.inria.fr)

- based on the Curry-Howard isomorphism (equivalence between proofs and λ-terms)
- few automations
- comprehensive libraries, including on $\mathbb{Z}$ and $\mathbb{R}$
- Coq kernel mechanically checks each step of each proof
- the method is to apply successively tactics (theorem application, rewriting, simplifications. . .) to transform or reduce the goal down to the hypotheses.
- the proof is handled starting from the conclusion.

(Courtesy Sylvie Boldo)

Polynomial models in Coq

Some simplifications compared to the algorithms in COSY:

- bounds on roundoff errors only use $|(a \diamond b| - RN(a \diamond b)| \leq (RU(a \diamond b) - RD(a \diamond b)/2$ and not $|(a \diamond b| - RN(a \diamond b)| \leq \mathrm{ulp}(1)/2 * \max(|a|, |b|)$
- polynomial multiplications are performed exactly (with high degrees) and then swept: simpler proofs but less efficient algorithms
- no threshold to avoid underflow, to keep sparse polynomials, no automated sweeping

Floating-point arithmetic in Coq

Simplified model for $\mathbb{F}$:

- no bounds on the exponents (ie. no overflow nor underflow),
- no ∞, no NaN.

Quarrels of experts: should $\mathbb{R}$ be classical, intuitionistic, ...
Main difference: whether a program can be extracted or not from the proof: not an issue here.

Conclusion

Personally, I am proud that the proofs I made for COSY were accepted reasonably easily by Coq (and by Milad Niqui, ie. they were also understandable by a human).

No error detected.

Future extensions

- model of floating-point numbers: why not use a more realistic model? why not use the library for interval arithmetic based on FP arithmetic? Cf. Flocq (`http://flocq.gforge.inria.fr/`).
- algorithms implemented in Ariadne: more operations (weak differentiation, anti-differentiation) and more elaborated algorithms (Newton, integration of ODEs, simulation of hybrid systems).

Conclusion

Personally, I am proud that the proofs I made for COSY were accepted reasonably easily by Coq (and by Milad Niqui, ie. they were also understandable by a human).
No error detected.

Future extensions

- model of floating-point numbers: why not use a more realistic model? why not use the library for interval arithmetic based on FP arithmetic? Cf. Flocq (`http://flocq.gforge.inria.fr/`).
- algorithms implemented in Ariadne: more operations (weak differentiation, anti-differentiation) and more elaborated algorithms (Newton, integration of ODEs, simulation of hybrid systems).

Introduction to IEEE-754 floating-point arithmetic

Nathalie Revol
INRIA – LIP - ENS de Lyon – France

TMW VII, 14-17 December 2011, Key West

From $\mathbb{R}$ to $\mathbb{F}$

Representing real numbers of a computer using a finite number of bits to approximate them.
For efficiency issues, a finite and fixed number of bits is preferred (32 or 64).

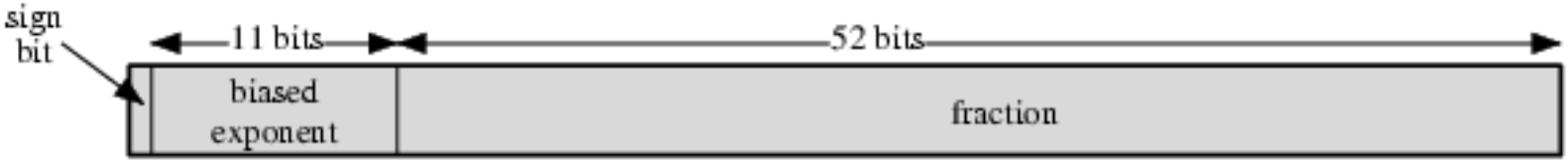
From $\mathbb{R}$ to $\mathbb{F}$

IEEE-754 representation:

- symmetry around 0, being able to represent a number and its opposite: 1 bit for the sign
- dynamics: floating-point representation (ie $m \times 10^e$ or $m \times 2^e$)
- symmetry wrt multiplication, being able to represent a number and its inverse (roughly speaking): the exponent range is symmetric around 0
- hidden bit: the first bit of the mantissa is a "1" (for normal representations) and does not need to be stored
- other considerations (precision)...

IEEE-754 representation for floating-point numbers

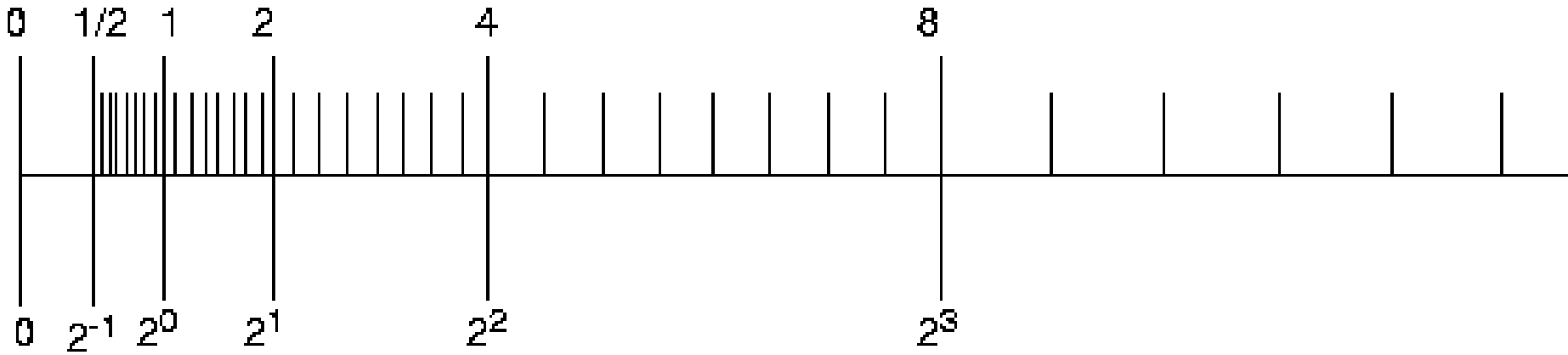
(a) Single format



(b) Double format

IEEE-754: underflow and subnormals

Binary Representation:



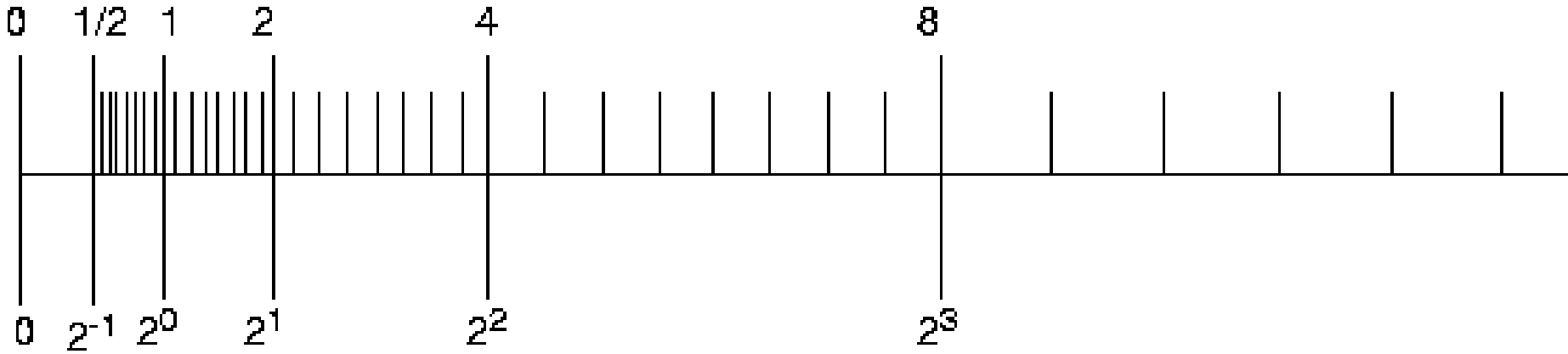
When the exponent is minimal, the rule for the hidden bit does not apply any more: the first bit is explicitely represented. Such numbers are called **subnormals**.

Roundoff error is no more relative but absolute for subnormals.

Specific (and delicate) handling of subnormals in proofs.

IEEE-754: underflow and subnormals

Binary Representation:



When the exponent is minimal, the rule for the hidden bit does not apply any more: the first bit is explicitely represented. Such numbers are called **subnormals**.

Roundoff error is no more relative but absolute for subnormals.

Specific (and delicate) handling of subnormals in proofs.

IEEE-754 representation: special values ∞ and NaN

Remark: 0 has two representations in $\mathbb{F}$: $+0$ and -0.

What happens for MAXFLOAT + MAXFLOAT?
Answer: $+\infty$ and $-\infty$ belong to $\mathbb{F}$.
And thus $1/(+0) = +\infty$ whereas $1/(-0) = -\infty$.

What happens for $0/0$?
Answer: this is an invalid operation, the result is NaN (Not a Number).

IEEE-754 representation: special values ∞ and NaN

Remark: 0 has two representations in $\mathbb{F}$: $+0$ and -0.

What happens for MAXFLOAT + MAXFLOAT?
Answer: $+\infty$ and $-\infty$ belong to $\mathbb{F}$.
And thus $1/(+0) = +\infty$ whereas $1/(-0) = -\infty$.

What happens for $0/0$?
Answer: this is an invalid operation, the result is NaN (Not a Number).

IEEE-754 representation: special values ∞ and NaN

Remark: 0 has two representations in $\mathbb{F}$: $+0$ and -0.

What happens for MAXFLOAT + MAXFLOAT?
Answer: $+\infty$ and $-\infty$ belong to $\mathbb{F}$.
And thus $1/(+0) = +\infty$ whereas $1/(-0) = -\infty$.

What happens for $0/0$?
Answer: this is an invalid operation, the result is NaN (Not a Number).

IEEE-754 representation: special values ∞ and NaN

Remark: 0 has two representations in $\mathbb{F}$: $+0$ and -0.

What happens for MAXFLOAT + MAXFLOAT?
Answer: $+\infty$ and $-\infty$ belong to $\mathbb{F}$.
And thus $1/(+0) = +\infty$ whereas $1/(-0) = -\infty$.

What happens for $0/0$?
Answer: this is an invalid operation, the result is NaN (Not a Number).

IEEE-754: specification of the arithmetic operations

To get

- the best possible accuracy
- reproducible results
- well-specified results
- ...

$a \diamond b$ should return the result as if it were computed exactly and then rounded: this is called *correct rounding*. The operation $\diamond$ can be $+$, $-$, $*$, $/$ or $\sqrt{}$.

Four rounding modes:

- rounding upwards,
- rounding downwards,
- rounding to zero,
- rounding to nearest (tie to even).

IEEE-754: specification of the arithmetic operations

Implementation issue: 3 bits suffice to get correct rounding.

Exact result: if the result is representable using floating-point arithmetic, then this exact result will be returned.

IEEE-754: other issues

Elementary functions (exp, log, sin, atan, cosh, ...): correct rounding is recommended (but not required).

Radix can be 2 or 10.

IEEE-754: properties on the error

With rounding to nearest.

If we compute using FP arithmetic

$$z := \frac{x}{\sqrt{x^2 + y^2}},$$

can we prove that $-1 \leq z \leq 1$?

Can we compute $t = \sqrt{1 - z^2}$ without getting a NaN?

The answer is **yes**.

IEEE-754: properties on the error

With rounding to nearest.

If we compute using FP arithmetic

$$z := \frac{x}{\sqrt{x^2 + y^2}},$$

can we prove that $-1 \leq z \leq 1$?

Can we compute $t = \sqrt{1 - z^2}$ without getting a NaN?

The answer is **yes**.

IEEE-754: properties on the error

Relative error $\leq$ 1 ulp or 1/2 ulp.
ulp (unit in the last place): it corresponds to the weight of the last bit of the mantissa.
In IEEE-754 double precision,

- with rounding to nearest, the relative error of one operation is $\leq 1/2$ ulp,
- with other rounding modes, the relative error of one operation is ≤ 1 ulp,

provided no overflow nor underflow occurs.

IEEE-754: properties on the error

Strictly between x and $x(1 - \mathrm{ulp}(x))$, there can be 0 or 1 floating-point number,
there is 1 if x is a power of 2.

Changes of binades must be handled with care in proofs.

IEEE-754: properties on the error

Strictly between x and $x(1 - \mathrm{ulp}(x))$, there can be 0 or 1 floating-point number,
there is 1 if x is a power of 2.

Changes of binades must be handled with care in proofs.

IEEE-754: properties on the error

Sterbenz lemma:

Let a and b be two positive floating-point numbers. If

$$\frac{a}{2} \leq b \leq 2a$$

then $a - b = a \ominus b$.

In other words, $a - b$ is exactly representable in floating-point arithmetic.

IEEE-754: properties on the error

Theorem:
for any pair (x, y) of FP numbers and for rounding to nearest, there exists FP numbers r_+ and r_- such that

$$\begin{aligned} r_+ &= (x + y) - (x \oplus y) \\ r_- &= (x - y) - (x \ominus y) \end{aligned}$$

Furthermore, r_+ and r_- can be computed using FP operations.

IEEE-754: properties on the error

Why with "rounding to nearest" only?

Counterexample for directed rounding

with basis 2 and at least $p > 4$ bits of mantissa, let's take

$$\begin{aligned} x &= -(2^{2p} + 2^{p+1}) \\ y &= 2^p - 3 \end{aligned}$$

then we have

$$\begin{aligned} x + y &= -2^{2p} - 2^p - 3 \\ x \oplus y &= -2^{2p} \\ (x + y) - (x \oplus y) &= -2^p - 3 = -(2^p + 3) \quad \text{not representable.} \end{aligned}$$

IEEE-754: computing the roundoff error

Let x and y be two normal FP numbers such that $|x| \geq |y|$ and the rounding mode be to nearest. Let also assume that $x \oplus y$ does not overflow.

$$\begin{array}{lrcl} \text{Algo Fast2Sum:} & s & = & x \oplus y \\ & z & = & s \ominus x \\ & r & = & y \ominus z \end{array}$$

The mathematical equality holds:

$$s + r = x + y$$

i.e. r is the roundoff error on the addition of x and y.
Beware of the optimizations done by your compiler...

IEEE-754: computing the roundoff error

The roundoff error can be computed without test (to compare x and y) at the expense of more arithmetic operations.
The roundoff error of a multiplication can also be computed (either using Dekker's algorithm or an FMA).

This means that it is possible to compute using FP arithmetic and twice the computing precision: in **double-double** arithmetic, each number is represented as the sum of two FP numbers (but the FP addition is not performed).

To conclude

IEEE-754 standard for floating-point arithmetic

- specifies the formats and the behaviour of the operations,
- makes it possible to bound roundoff errors, to track them during computations (cf. N. Higham: *Accuracy and Stability of Numerical Algorithms*, SIAM 2002),
- makes it possible to compute the roundoff error,
- makes it possible to compute in higher precision,
- makes it possible to establish proofs on the quality of the results...
  but this is tedious and error-prone (many cases to handle: possibility of overflow, underflow, change of binade...).

Formal Proofs of Polynomial Models

Nathalie Revol
INRIA – LIP - ENS de Lyon – France

14 December 2011

Polynomial models: definition

Notations: $x^{\alpha} = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_v^{\alpha_v}$.
All variables belong to $[-1, 1]$.

Polynomial model: it is a pair $< p, \mathbf{I} >$ where p is a polynomial and $\mathbf{I}$ is an interval.

$< p, \mathbf{I} >$ **represents the function** $f : \mathbb{R}^v \to \mathbb{R}$ if $\forall x \in [-1, 1]^v$, $f(x) - p(x) \in \mathbf{I}$.

Polynomial models: arithmetic

Operations can be defined between polynomial models: addition, subtraction, multiplication by a scalar, multiplication, composition...

They use the corresponding operations on polynomials and on intervals.
(Beware: the degree of the resulting polynomial may vary.)

Polynomial models: sweeping

To get a simpler representation, ie. a polynomial with less coefficients: **sweeping operation**.

The "lost" coefficients are accounted for in the interval remainder: I becomes $I + \sum[-|a_i|, |a_i|]$ where the sum is over swept terms.

Thus manipulated polynomials can keep constant degrees: higher degree terms are swept.

Polynomial models: implementation

To implement polynomial models and arithmetic on a computer: use of floating-point arithmetic to represent the coefficients and to perform arithmetic operations on them.

Floating-point operations are not exact
thus rounding errors have to be accounted for in the interval remainder.
How?

Polynomial models: implementation

Bounding the error of one operation:
if the operation is performed using rounding to nearest, then the error between the exact operation and the rounded-to-nearest operation verifies

$$a \diamond b - RN(a \diamond b) \leq (RU(a \diamond b) - RD(a \diamond b))/2$$

where RN, RU, RD stand for rounding to nearest, upward, downward.

(Again, the division by 2 should be performed using RU.)

Polynomial models: implementation

Example of the addition of two polynomial models

$< p, \mathbf{I} > + < q, \mathbf{J} >$
where $p = \sum_{i=0}^{m} a_i x^{\alpha_i}$ and $q = \sum_{j=0}^{n} b_j x^{\beta_j}$. Then

$$< p, \mathbf{I} > + < q, \mathbf{J} > = < \sum_{k=0}^{\max(m,n)} RN(a_k + b_k) x^{\alpha_k}), \mathbf{I} + \mathbf{J} + \sum_{k=0}^{\max(m,n)} (RU(a_k + b_k) - RD(a_k + b_k))/2 > .$$

Example of lemma used in COSY

Let a and b be two floating-point numbers. Then

$$|(a+b) - RN(a+b)| \leq RN((\mathrm{ulp}(1)/2) * \max(|a|, |b|))$$

provided no overflow nor underflow occurs.

Example of lemma: proof

Case distinction:

- a and b have opposite signs;
- $a = b$;
- a and b are wide apart (assume $|b| < |a|$ and $\mathrm{ulp}(1)/2 \leq \beta \leq 1$);
- $b = (1 - \beta)a$ with $0 < \beta < \mathrm{ulp}(1)/2$ (assume $|b| < |a|$), case distinction again, with 2-3 possible values for b in each case:
  - rounding mode to nearest or not?
  - a is a power of 2 or not?
  - $a-$ (the floating-point immediately closer to 0 than a) is a power or 2 or not?

Example of proof

	rounding to nearest (even) $\varepsilon_m = u$		other rounding modes $\varepsilon_m = 2u$		
	b	error		b	error
$a = 2^t$	a^-	$ua/2 \leq \varepsilon_m \otimes a$		a^-	$ua/2 \leq \varepsilon_m \otimes a$
	a^{--}	$0 \leq \varepsilon_m \otimes a$		a^{--}	$0 \leq \varepsilon_m \otimes a$
				a^{---}	$ua/2 \leq \varepsilon_m \otimes a$
$2^t < a < 2^{t+1}$	a^-	$ua \leq \varepsilon_m \otimes a$	$a^- = 2^t$	a^-	$ua^- \leq \varepsilon_m \otimes a$
				a^{--}	$3ua^-/2 \leq \varepsilon_m \otimes a$
				a^{---}	$0 \leq \varepsilon_m \otimes a$
			$2^t < a^-$	a^-	$u2^t \leq \varepsilon_m \otimes a$
				a^{--}	$0 \leq \varepsilon_m \otimes a$

Do you trust this proof?

Example of proof

	rounding to nearest (even) $\varepsilon_m = u$		other rounding modes $\varepsilon_m = 2u$		
	b	error		b	error
$a = 2^t$	a^-	$ua/2 \leq \varepsilon_m \otimes a$		a^-	$ua/2 \leq \varepsilon_m \otimes a$
	a^{--}	$0 \leq \varepsilon_m \otimes a$		a^{--}	$0 \leq \varepsilon_m \otimes a$
				a^{---}	$ua/2 \leq \varepsilon_m \otimes a$
$2^t < a < 2^{t+1}$	a^-	$ua \leq \varepsilon_m \otimes a$	$a^- = 2^t$	a^-	$ua^- \leq \varepsilon_m \otimes a$
				a^{--}	$3ua^-/2 \leq \varepsilon_m \otimes a$
				a^{---}	$0 \leq \varepsilon_m \otimes a$
			$2^t < a^-$	a^-	$u2^t \leq \varepsilon_m \otimes a$
				a^{--}	$0 \leq \varepsilon_m \otimes a$

Do you trust this proof?

Proof checker

What is it?

It is as if a very pernickety reader checks your proof.

As it is not human, no chance that it forgets one case, nor that it admits an implicit assumption.

But sometimes this can be tedious: you have to explain to it that $2_{\mathbb{N}} = 2_{\mathbb{R}}$... and many such things.

Proof checker

What is it?

It is as if a very pernickety reader checks your proof.

As it is not human, no chance that it forgets one case, nor that it admits an implicit assumption.

But sometimes this can be tedious: you have to explain to it that $2_{\mathbb{N}} = 2_{\mathbb{R}}$... and many such things.

Proof checker

The proof is checked in its deep details until the computer agrees with it.
Often, use of formal proof checkers, meaning programs that only check a proof (they may also generate easy demonstrations).
Therefore the checker is a very short program (de Bruijn criteria : the correctness of the system as a whole depends on the correctness of a very small kernel).

Why use a proof checker?

Even if polynomial models already provide validated computations...
this provides one extra level of confidence.

Proof checker

Which proof checker? The Coq proof assistant (http://coq.inria.fr)

- based on the Curry-Howard isomorphism (equivalence between proofs and λ-terms)
- few automations
- comprehensive libraries, including on $\mathbb{Z}$ and $\mathbb{R}$
- Coq kernel mechanically checks each step of each proof
- the method is to apply successively tactics (theorem application, rewriting, simplifications. . .) to transform or reduce the goal down to the hypotheses.
- the proof is handled starting from the conclusion.

Polynomial models in Coq

Some simplifications compared to the algorithms in COSY:

- bounds on roundoff errors only use
  $|(a \diamond b| - RN(a \diamond b)| \leq (RU(a \diamond b) - RD(a \diamond b)/2$ and not
  $|(a \diamond b| - RN(a \diamond b)| \leq \mathrm{ulp}(1)/2 * \max(|a|, |b|)$
- polynomial multiplications are performed exactly (with high degrees) and then swept: simpler proofs but less efficient algorithms
- no threshold to avoid underflow, to keep sparse polynomials, no automated sweeping

Floating-point arithmetic in Coq

Simplified model for $\mathbb{F}$:

- no bounds on the exponents (ie. no overflow nor underflow),
- no ∞, no NaN.

Quarrels of experts: should $\mathbb{R}$ be classical, intuitionistic, ...
Main difference: whether a program can be extracted or not from the proof: not an issue here.

Conclusion

Personally, I am proud that the proofs I made for COSY were accepted reasonably easily by Coq (and by Milad Niqui, ie. they were also understandable by a human).

No error detected.

Future extensions

- model of floating-point numbers: why not use a more realistic model? why not use the library for interval arithmetic based on FP arithmetic? Cf. Flocq (`http://flocq.gforge.inria.fr/`).
- algorithms implemented in Ariadne: more operations (weak differentiation, anti-differentiation) and more elaborated algorithms (Newton, integration of ODEs, simulation of hybrid systems).

Conclusion

Personally, I am proud that the proofs I made for COSY were accepted reasonably easily by Coq (and by Milad Niqui, ie. they were also understandable by a human).
No error detected.

Future extensions

- model of floating-point numbers: why not use a more realistic model? why not use the library for interval arithmetic based on FP arithmetic? Cf. Flocq (`http://flocq.gforge.inria.fr/`).
- algorithms implemented in Ariadne: more operations (weak differentiation, anti-differentiation) and more elaborated algorithms (Newton, integration of ODEs, simulation of hybrid systems).