

CSCC37 Floating Point Number Notes

1. Introduction:

- Can the following numbers be represented exactly on a computer:
 - $\pi \rightarrow$ No, because it goes on forever.
 - $\sqrt{2} \rightarrow$ No, because it goes on forever
 - $\frac{1}{10} \rightarrow$ No, because there are conversion issues.

2. Representation of Non-negative Integers:

- Decimal (Base 10) System:
- $(350)_{10} = 3 \times 10^2 + 5 \times 10^1 + 0 \times 10^0$

- Binary (Base 2) System:

$$(101011110)_2 = 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$
$$= (350)_{10}$$

- Hexadecimal (Base 16) System:

- Has 0, 1, ..., 9, A, B, C, D, E, F

$$(8FA)_{16} = 8 \times 16^2 + F \times 16^1 + A \times 16^0$$

F is the 15th digit
digit, so we substitute 15 for it.
A is the 10th digit
so we substitute 10 for it.

$$= 8 \times 16^2 + 15 \times 16^1 + 10 \times 16^0$$
$$= (2298)_{10}$$

- Given a base b system, where $b > 0$ and $b \in \mathbb{Z}$, suppose $x = (d_n d_{n-1} \dots d_0)_b$. Then, $x = d_n \cdot b^n + d_{n-1} \cdot b^{n-1} + \dots + d_0 \cdot b^0$, where $x \geq 0$, $x \in \mathbb{N}$, $0 \leq d_i < b$, $i = 0, 1, \dots, n$.

- Converting from base b to base 10:
- Suppose we have $x = (d_n d_{n-1} \dots d_0)_b$. To convert x to base 10, simply do $d_n b^n + d_{n-1} b^{n-1} + \dots + d_0 b^0$.

E.g. 1 Convert $(235)_8$ to base 10.

Soln:

$$\begin{aligned}(235)_8 &= 2 \cdot 8^2 + 3 \cdot 8^1 + 5 \cdot 8^0 \\ &= 128 + 24 + 5 \\ &= (157)_{10}\end{aligned}$$

E.g. 2 Convert $(1011)_2$ to base 10.

Soln:

$$\begin{aligned}(1011)_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 8 + 0 + 2 + 1 \\ &= (11)_{10}\end{aligned}$$

- Converting from base 10 to base b :

E.g. 3 Convert $(350)_{10}$ to base 2

Soln:

Numerator	Denominator	Quotient	Remainder
350	2	175	0
175	2	87	1
87	2	43	1
43	2	21	1
21	2	10	1
10	2	5	0
5	2	2	1
2	2	1	0
1	2	0	1

$$(350)_{10} = (10101110)_2$$

Read
bottom
to
up

- This method/technique works for any base.
- We stop when the quotient = 0. (It will always reach 0 at some point.)
- For the conversion, write the values in the remainder column from bottom to top.
- This conversion is safe except for overflow.

E.g. 4 Convert $(110)_{10}$ to base 16.

Soln:

Numerator	Denominator	Quotient	Remainder
110	16	6	14
6	16	0	6

Since the 14th digit in hexadecimal is E,
 $(110)_{10} = (6E)_{16}$

E.g. 5 Convert $(140)_{10}$ to base 8.

Soln:

Numerator	Denominator	Quotient	Remainder
140	8	17	4
17	8	2	1
2	8	0	2

$$(140)_{10} = (214)_8$$

3. Representation of Reals:

- If $x \in R$, then $x = \pm(x_I \cdot x_F)_b$ where
 $x_I = \text{Integral part}$ and $x_F = \text{Fractional part}$.

$$\begin{aligned} x &= \pm(x_I \cdot x_F)_b \\ &= \pm(d_n \dots d_1 d_0 \dots)_b \end{aligned}$$

Note: The sign, + / -, is 1 bit, 0 or 1.

Note: x_I is a non-negative integer.

Note: x_F can be infinite.

$$\begin{aligned} \text{E.g. } (0.77\dots)_{10} &= (0.\bar{7})_{10} \\ &= 7 \times 10^{-1} + 7 \times 10^{-2} + \dots \end{aligned}$$

- For a binary system, suppose we have $(.00011001\dots)_2$. This is equivalent to $(.0\overline{001})_2$ which equals to $0 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-5} + \dots$

- Given a base b system, suppose we have

$$x_F = (.d_1 d_2 \dots)_b. \text{ Then:}$$

$$x_F = d_1 b^{-1} + d_2 b^{-2} + \dots$$

$$= \sum_{i=1}^{\infty} d_i b^{-i}$$

Note: A terminating binary fraction with n digits always has a terminating decimal representation.

However, in base 3, this isn't true.

$$\text{E.g. } (0.1)_3 = 1 \times 3^{-1}$$

$$= \frac{1}{3} \leftarrow \text{Does not terminate}$$

- Converting reals in base 10 to base b :

E.g. 6 Convert $(.625)_{10}$ to base 2.

Soln:

Multiplier	Base	Product	Integral	Fraction
.625	2	1.25	1	0.25
0.25	2	0.5	0	0.5
0.5	2	1.0	1	0

$$(0.625)_{10} = (.101)_2$$

- We stop when the fraction column hits 0.

Note: Sometimes, it may never hit 0.

- We read the integral column top down to get the corresponding base b value.

Note: A terminating decimal fraction may not have a terminating binary fraction.

E.g. 7 Convert $(.1)_{10}$ to base 2

Soln:

Multiplier	Base	Product	Integral	Fraction
0.1	2	0.2	0	0.2
0.2	2	0.4	0	0.4
0.4	2	0.8	0	0.8
0.8	2	1.6	1	0.6
0.6	2	1.2	1	0.2
0.2	2	0.4	0	0.4
0.4	2	0.8	0	0.8
0.8	2	1.6	1	0.6
0.6	2	1.2	1	0.2

Repeats

The cycle is 0011.

$$\text{Hence, } (.1)_{10} = (.0\overline{0011})_2.$$

Hence, $(.1)_{10}$ converts to a non-terminating binary fraction.

Recall: In the introduction section (pg 1) we said that 0.1 cannot be represented exactly on a computer. This is why.

E.g. 8 Convert $(72.6)_{10}$ to base 3

Soln:

To solve this, we need to split the integral and fraction parts.

$(72)_{10}$ to base 3:

Numerator	Denominator	Quotient	Remainder
72	3	24	0
24	3	8	0
8	3	2	2
2	3	0	2

$$(72)_{10} = (2200)_3$$

$(.6)_{10}$ to base 3:

Multiplier	Base	Product	Integral	Fraction
0.6	3	1.8	1	0.8
0.8	3	2.4	2	0.4
0.4	3	1.2	1	0.2
0.2	3	0.6	0	0.6

I will stop here as it will repeat.

$$(0.6)_{10} = (\overline{1210})_3$$

$$(72.6)_{10} = (2200.\overline{1210})_3$$

E.g. 9 Convert $(86.4)_{10}$ to base 4

Soln:

$(86)_{10}$ to base 4:

Numerator	Denominator	Quotient	Remainder
86	4	21	2
21	4	5	1
5	4	1	1
1	4	0	1

$$(86)_{10} = (1112)_4$$

$(.4)_{10}$ to base 4:

Multiplier	Base	Product	Integral	Fraction
0.4	4	1.6	1	0.6
0.6	4	2.4	2	0.4

I will stop here as it is clear that it will repeat.

$$(0.4)_{10} = (\overline{12})_4$$

$$\text{Hence, } (86.4)_{10} = (1112.\overline{12})_4$$

4. Machine Representation of Reals:

- Real numbers are represented in computers as **Floating Point Numbers (FPN)**.

- A FPN x in base b has the form:

$x = (F)_b \cdot b^{(e)_b}$, where F is the fraction part and has the form $F = \pm (.d_1.d_2...d_t)_b$ and e is the exponent and has the form $e = \pm (c_s c_{s-1} ... c_1)_b$.

I.e.

$F = \pm (.d_1.d_2...d_t)_b$ is the **fraction part**.

$e = \pm (c_s c_{s-1} ... c_1)_b$ is the **exponent**.

Note: Another term for F is the **mantissa**.

- On a 64-bit computer, 1 bit is used for the sign, 11 bits are used for the exponent and 52 bits are used for the mantissa.

- A FPN is **normalized** if $d_1 \neq 0$ unless $d_1 = d_2 = \dots = d_t = 0$.

I.e. A FPN is normalized if $d_1 \neq 0$ unless $d_i = 0$ $\forall i = 0, 1, \dots, t$.

- There's 2 reasons why we want normalized FPNS:

1. Uniqueness
2. Storage efficiency / Better storage

E.g. 10 Suppose we are using a 32-bit computer where 1 bit is used for the sign, 8 bits are used for the exponent and 23 bits are used for the mantissa. Represent $(-53.5)_{10}$ in base 2 and write the FPN equivalent.

Soln:

Converting $(53.5)_{10}$ to base 2:

a) Converting $(53)_{10}$ to base 2:

Numerator	Denominator	Quotient	Remainder
53	2	26	1
26	2	13	0
13	2	6	1
6	2	3	0
3	2	1	1
1	2	0	1

$$\text{Hence } (53)_{10} = 110101$$

b) Converting $(0.5)_{10}$ to base 2:

Multiplier	Base	Product	Integral	Fraction
0.5	2	1.0	1	0

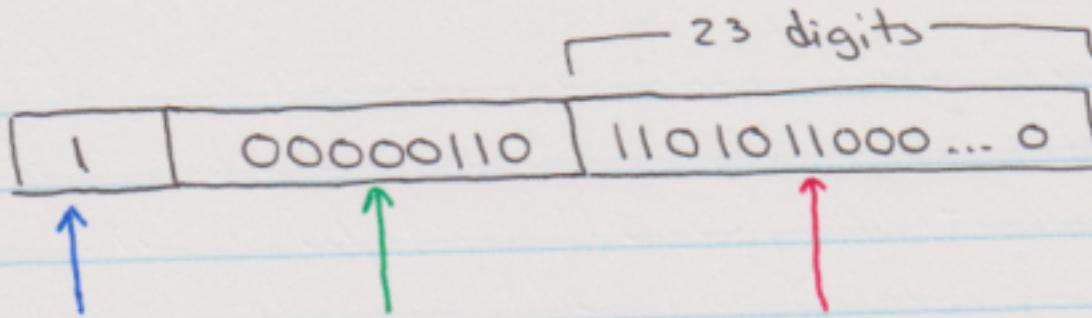
$$\text{Hence, } (0.5)_{10} = 0.1_2.$$

$$\therefore (53.5)_{10} = (110101.1)_2 \rightarrow (-53.5)_{10} = (-110101.1)_2$$

Writing the FPN equivalent of $(-110101.1)_2$:

$$(-110101.1)_2 \rightarrow (-0.110101)_2 \cdot 2^{(110)_2}$$

Note: We had to shift the decimal point 6 spots to the left, so we multiply (-0.110101) by 2^6 , but 6 is in base 10, so we have to convert it to base 2, which is 110.



One bit for the sign. 8 bits for the exponent. 23 bits for the mantissa.

- Significant digits of a non-zero FPN are the digits following and including the first non-zero digit.

Note: In a normalized FPN, all digits of the mantissa are significant.

- For the purposes of our course, the absolute value of the mantissa is always fractional.
I.e. $0 \leq |\text{mantissa}| < 1$
- The exponent is also limited. It can only have s digits at most.
- The value of the exponent is bound by $-M$ and M where $M = \underbrace{(aa\dots a)}_s_b$ where $a = b-1$.

I.e. $-M \leq e \leq M$ where $M = \underbrace{(aa\dots a)}_s_b$ and $a = b-1$.

Consider example 10. In that example, $b=2$ and $s=8$.
In that example, $e \leq 11111111$ and $-11111111 \leq e$.
I.e. $-11111111 \leq e \leq 11111111$

- The absolute value of the largest FPN defined by this system is

$$\underbrace{(aa\dots a)_b \cdot b}_{t} \underbrace{\overbrace{s}}_{\substack{(aa\dots a)_b \\ s}}$$

where $a = b - 1$.

Suppose $b = 2$, $s = 3$, $t = 4$. The largest FPN is

$$(1111)_2 \cdot 2^{\underbrace{(111)_2}_{s}}$$

- The smallest non-zero, normalized FPN is

$$\underbrace{(.10\dots 0)_b \cdot b}_{t} \underbrace{\overbrace{s}}_{\substack{-(a\dots a)_b \\ s}}$$

- The smallest non-zero FPN is $\underbrace{(.00\dots 1)_b \cdot b}_{t} \underbrace{\overbrace{s}}_{\substack{-(a\dots a)_b \\ s}}$.

Non-normalized FPNs allow us to get very close to 0.

- $R_b(t,s)$ denotes the set of all FPN with base b , t digit mantissa and s digit exponent.

- **Overflow or underflow** occurs whenever a non-zero FPN with abs value \equiv outside the ranges must be stored on a computer.

When the number gets too close to 0, it's **underflowing**. When the number gets too large, in the positive or negative direction, it's **overflowing**.

E.g. 11 Suppose we have 8-bit signed ints. The range of representable ints start at -128 and ends at 127.

If we do $127 + 1$, it causes an overflow as the value is outside of the given range.

Likewise, if we do $-128 - 1$, it causes an overflow.

Now, suppose that the exponent part can represent values from -127 to 127. Then, any number with abs value less than 2^{-127} may cause underflow.

- The number of normalized FPN is $2(B-1)B^{t-1}(U-L+1)+1$ where
 - The 2 is used for the 2 possible signs.
 - The $(B-1)$ is used to represent the number of possible values the first digit can have.
 - Recall in a normalized FPN, the leading digit can't be 0.
 - The B^{t-1} is used to represent the number of possible values each digit after the first can have.
 - The $U-L+1$ is used to represent the number of values the exponent can have. Note: $\begin{matrix} U=M \\ L=-M \end{matrix}$
 - The $+1$ is used in case the number is 0.
- The smallest positive normalized FPN, also called the **underflow level**, is B^{-M} , which has a 1 for the first digit of the mantissa and 0 elsewhere. Earlier, we defined this to be $(.10\dots 0)_b \cdot b^{-(a\dots 0)}$. This is equal to

$$\begin{aligned} & 1 \times b^{-(a\dots 0)} \\ & = b^{-(a\dots 0)} \\ & = b^{-M} \end{aligned}$$

- The largest FPN, also known as the **overflow level**, is $B^{n+1}(1-B^{-t})$. Earlier, we defined this to be $(.a...a)_b \cdot b^{(a...a)_b}$.
 - $R_b(t,s)$ is finite while \mathbb{R} is infinite. Furthermore, \mathbb{R} is compact while $R_b(t,s)$ isn't. This means that between any 2 real numbers, there is an infinite number of reals in between.
 - A real number $x = \pm(x_I, x_F)_b$
 $= \pm(d_nd_{n-1}...d_0.d_{-1}d_{-2}...)_b$
- can be represented in $R_b(t,s)$ by the following algo:
1. Normalize the mantissa:
 - Shift the decimal point to the left of d_n .
 - $x = \pm(d_nd_{n-1}...d_0.d_{-1}...)_b \rightarrow \pm(.D_0D_1...)_b \cdot b^{n+1}$
 2. Round or Chop off the Mantissa:
 - a) Chopping: Chops after t digits of the mantissa.
 - b) Rounding: Chop off after digit t then round D_t up if $D_{t+1} \geq b/2$ and down if $D_{t+1} < b/2$.

Note: Another, possibly more efficient, technique of rounding is to add $b/2$ to digit D_{t+1} and then chop after D_t .

We convert x to FPN with the following notation:
 $x \in \mathbb{R} \rightarrow \text{FL}(x) \in R_b(t,s)$.

E.g. Consider $\text{FL}(2/3) \in R_{10}(2,4)$.

It equals $\begin{cases} 0.66, & \text{if chopped} \\ 0.67, & \text{if rounded} \end{cases}$

- Round off error is the difference between $x \in \mathbb{R}$ and $FL(x) \in R_b(t,s)$.

It is usually measured relative to x as $\frac{x - FL(x)}{x} = d$

or $FL(x) = x(1-d)$ where d is the relative round off.

$$\text{Absolute Error (AE)} = x - FL(x)$$

$$\text{Relative Error (RE)} = \frac{x - FL(x)}{x}$$

We can bound d independently of x .

$d < b^{1-t}$ for chopped, normalized FPNS.

$|d| < \frac{b^{1-t}}{2}$ for rounding normalized FPNS.

5. Machine Arithmetic:

- Let $x, y \in \mathbb{R}$ and $FL(x), FL(y) \in R_b(t,s)$.

Consider $\circ \in \{+, -, \times, /\}$. I.e. \circ is an operation.

$$x \circ y \approx FL(FL(x) \circ FL(y))$$

E.g. 12 In $R_{10}(2,4)$, let $x=2$ and $y=0.0000058$.

Suppose we do $x+y$.

$$\begin{aligned} x+y &= FL(FL(x) + FL(y)) \\ &= FL(0.20 \cdot 10^1 + 0.58 \cdot 10^{-5}) \\ &= FL(0.20000058 \cdot 10^1) \\ &= 0.20 \cdot 10^1 \end{aligned}$$

6. Machine Precision / Machine Epsilon:

- Defined as the smallest non-normalized FPN
 eps s.t. $1 + \text{eps} > 1$. Eps is referred to as machine epsilon.

$$-\text{eps} = \begin{cases} b^{1-t} & \text{if chopping} \\ \frac{b^{1-t}}{2} & \text{if rounding} \end{cases}$$

Hence, $0 \leq d \leq \text{eps}$ for chopping and
 $|d| \leq \text{eps}$ for rounding

E.g. 13

Let $\circ \in \{+, -, \times, /\}$

Let $x, y \in \mathbb{R}$ and $FL(x) = x(1-d)$, $FL(y) = y(1-d)$

Find / Calculate the error bound for $x \circ y$.

Soln:

$$\begin{aligned} x \circ y &= FL(FL(x) \cdot FL(y)) \\ &= FL(x(1-d_x) \cdot y(1-d_y)) \\ &= x(1-d_x) \cdot y(1-d_y) \cdot (1-d_{xy}) \\ &= xy ((1-d_x)(1-d_y)(1-d_{xy})) \\ &= xy ((1-d_x-d_y + d_x d_y)(1-d_{xy})) \\ &= xy (1 - d_{xy} - d_x + d_x d_{xy} - d_y + d_y d_{xy} + d_x d_y - d_x d_y d_{xy}) \\ &\approx xy (1 - d_x - d_y - d_{xy}) \\ &= xy (1 - d.) \end{aligned}$$

Since $|d_x| \leq \text{eps}$ and $|d_y| \leq \text{eps}$ and $|d_{xy}| \leq \text{eps}$,
 $|d| \leq 3 \cdot \text{eps}$. Note that this is the worst case scenario.

This is a good approximation because since each d_i is a small fraction, multiplying them ^{with each other} will result in values so small, they are much lower than eps . Hence, we can remove all instances of 2 or more d_i 's times each other.

E.g. 14

Let $x, y \in \mathbb{R}$ and $FL(x) = x(1-d)$, $FL(y) = y(1-d)$.
 Calculate the error bound for $x+y$.

Soln:

$$\begin{aligned}
 x+y &= FL(FL(x) + FL(y)) \\
 &= [x(1-dx) + y(1-dy)](1-dxy) \\
 &= x(1-dx)(1-dxy) + y(1-dy)(1-dxy) \\
 &= x(1-dxy - dx + dxdxy) + y(1-dxy - dy + dydxy) \\
 &\approx x(1-dxy - dx) + y(1-dxy - dy) \\
 &= (x+y) \left(1 - \frac{x(dx + dxy)}{x+y} - \frac{y(dy + dxy)}{x+y} \right) \\
 &= (x+y)(1-d+)
 \end{aligned}$$

$$|d+| \leq \left| \frac{x}{x+y} \right| \cdot 2\text{eps} + \left| \frac{y}{x+y} \right| \cdot 2\text{eps}$$

$$= \frac{|x| + |y|}{|x+y|} \cdot 2\text{eps}$$

$$|d+| \leq \begin{cases} 2\text{eps}, & \text{when } x \text{ and } y \text{ have the same sign} \\ 2\text{eps} \cdot \frac{|x-y|}{|x+y|}, & \text{when } x \text{ and } y \text{ have the opposite signs.} \end{cases}$$

Note: As x approaches $-y$, the error bound approaches infinity. This is called **subtractive cancellation**.

E.g. 15 Consider $R_{10}(3,1)$ with rounding.
Compute $a^2 - 2ab + b^2$ with $a = 15.6$ and $b = 15.7$.

Soln:

$$FL(a) = 0.156 \cdot 10^2$$

$$FL(b) = 0.157 \cdot 10^2$$

$$FL(a^2) = FL(243.36) = +(0.243 \cdot 10^3)$$

$$FL(2ab) = FL(489.84) = +(0.490 \cdot 10^3)$$

$$FL(b^2) = FL(246.49) = +(0.246 \cdot 10^3)$$

$$\begin{aligned} FL(a^2 - 2ab + b^2) &= FL(243 - 490 + 246) \\ &= FL(-1) \\ &= -(0.100 \cdot 10^1) \end{aligned}$$

This is an issue because $a^2 - 2ab + b^2 = (a-b)^2$, which is positive. This is an example of subtractive cancellation.

7. Stability of Formulae:

- Used for algorithms.
- An algorithm is **stable** if the result it produces is relatively insensitive to perturbations resulting from approximations made during the computation.
- One way to deal with unstable algorithms is to use a different, more stable algorithm.

E.g. 16 Consider $1 - \cos x$.

When x approaches 0, $\cos x$ approaches 1 and the formula suffers from subtractive cancellation.

We can use an alternative formula.

$$1 - \cos x \left(\frac{1 + \cos x}{1 + \cos x} \right)$$

$$= \frac{1 - \cos^2 x}{1 + \cos x}$$

$$= \frac{\sin^2 x}{1 + \cos x}$$

This new formula is fine when x approaches 0 but not fine when x approaches π as $\cos(\pi) = -1$.

The original formula is fine when x approaches π .

Hence, depending on the value of x , choose a formula that doesn't suffer from subtractive cancellation.

use
$$\begin{cases} 1 - \cos x, & \text{if } x \text{ approaches } \pi \\ \frac{\sin^2 x}{1 + \cos x}, & \text{if } x \text{ approaches } 0 \end{cases}$$

8. Condition of Functions:

- Used for functions.
- A function is **well-conditioned** if a given relative change in the input data causes a reasonably proportionate relative change in the solution.
- A function is **ill-conditioned** if the relative change in the solution can be much larger than that in the input data.
- The condition of a function can be calculated as

$$\text{Cond}(f) = \frac{|\text{relative change in soln}|}{|\text{relative change in input data}|}$$

$$= \frac{|(f(\hat{x}) - f(x))/f(x)|}{|(\hat{x} - x)/x|}, \text{ where } \hat{x} \text{ is a point near } x$$

$$= \frac{|xf'(\hat{x})|}{|f(x)|}$$

$$\approx \frac{|xf'(x)|}{|f(x)|}$$

To calculate the conditioning of a function, we'll use
 $\text{cond}(f) = \frac{|x f'(x)|}{|f(x)|}$

function, f ,

- A f is ill-conditioned if $\text{cond}(f)$ is much larger than 1.

E.g. 17 Let $f(x) = \sqrt{x}$. Calculate $\text{cond}(f)$.

Soln:

$$\begin{aligned}\text{cond}(f) &= \frac{|x f'(x)|}{|f(x)|} \\ &= \frac{|x (\sqrt{x})'|}{|\sqrt{x}|} \\ &= \frac{|x|}{|2(\sqrt{x})(\sqrt{x})|} \\ &= \frac{|x|}{|2x|} \\ &= \frac{1}{2} \leftarrow \text{well conditioned}\end{aligned}$$

E.g. 18 Let $f(x) = \frac{10}{1-x^2}$. Calculate $\text{cond}(f)$.

Soln:

$$\begin{aligned}\text{cond}(f) &= \frac{|x f'(x)|}{|f(x)|} \\ &= \frac{\left| x \cdot \frac{20x}{(1-x^2)^2} \right|}{\left| \frac{10}{1-x^2} \right|} \\ &= \frac{|2x^2|}{|1-x^2|} \leftarrow \text{ill conditioned when } |x| \approx 1\end{aligned}$$

- If algo A is stable, then function f is well-defined. However, if f is well-defined, there's a chance we can't find a stable algo to compute it.