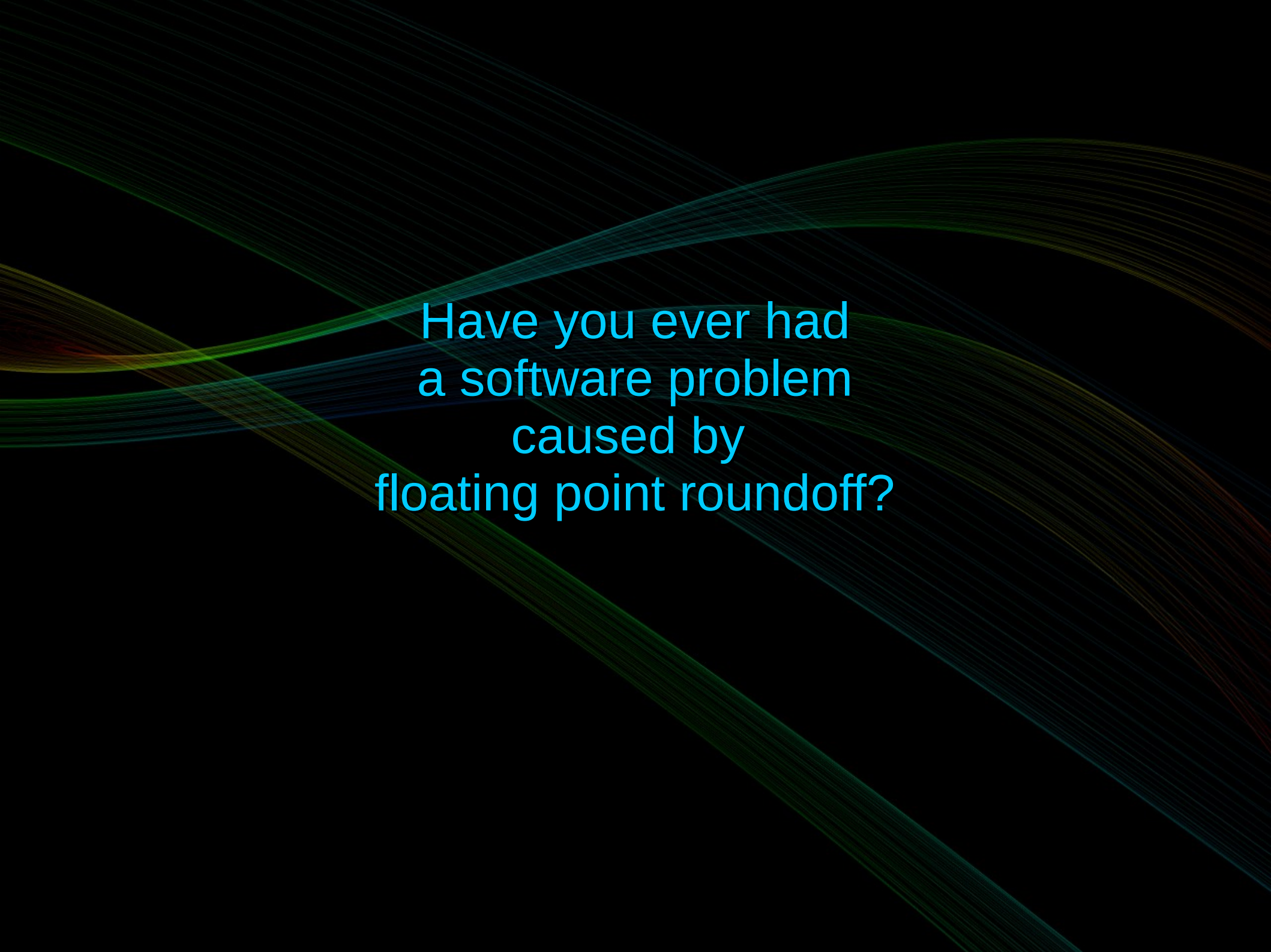


Exact Real Arithmetic in Tcl

Kevin B. Kenny

22nd Annual Tcl/Tk Conference
21 October 2015



Have you ever had
a software problem
caused by
floating point roundoff?

Were you using floating point
for currency?

Did you learn from that mistake?

You haven't had floating-point
precision problems?

How do you know?

Do you test your software
for the accuracy of
floating-point results?

*How do you know what
the right answers are?*

Floating point is full of horror stories

Many involve the precision
of intermediate results,
not the input data nor
the ultimate answers

Catastrophic loss of significance

Let's say that we have the equation

$$Ax^2 + Bx + C = 0$$

And we want to find x . Asking the nearest high-schooler for how to do it we get told,

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

Everyone knows that! Brahmagupta published it in A.D. 678! People have used that formula for 1500 years!

Catastrophic loss of significance

```
proc quad1 {a b c} {  
  set d [expr {sqrt($b*$b - 4.*$a*$c)}]  
  set r0 [expr {(-$b - $d) / (2. * $a)}]  
  set r1 [expr {(-$b + $d) / (2. * $a)}]  
  return [list $r0 $r1]  
}
```

```
quad1 1.0 200.0 -1.5e-12
```

```
→ -200.0 1.4210854715202004e-14
```

What?

The correct answer is approximately 7.5×10^{-15} !

(The answer: $-200. - \text{sqrt}(400.0000000000006)$ loses all but one bit of the significand.)

You're doing it wrong!

An experienced numerical analyst will tell you that the right way to use the quadratic formula is to write it in a different way:

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \quad x = \frac{2C}{-B \pm \sqrt{B^2 - 4AC}}$$

Experienced numerical analysts are expensive.
They can take a long time to solve your problem.
They still make mistakes and miss things.
How do you know when *they* have it right?

A popular idea: variable precision

Repeat calculations at different precisions

See what changes as precision increases: do we get the same answer?

But consider the problem of finding the limit of:

$$x_0 = 4.00$$

$$x_1 = 4.25$$

$$x_n = 108 - \frac{815 - \frac{1500}{x_{n-2}}}{x_{n-1}}$$

Variable precision: what happens?

#	float	double	exact
1	4.47059	4.47059	4.47059
2	4.64474	4.64474	4.64474
3	4.77053	4.77054	4.77054
4	4.85545	4.85570	4.85570
5	4.90575	4.91085	4.91085
6	4.84165	4.94554	4.94554
7	2.82180	4.96696	4.96696
8	-71.03029	4.98004	4.98004
9	111.99020	4.98791	4.98798
10	100.53401	4.99136	4.99277
11	100.02652	4.96746	4.99566
12	100.00133	4.42969	4.99739
13	100.00007	-7.81724	4.99843
14	100.00000	168.93917	4.99906
15	100.00000	102.03996	4.99944
16	100.00000	100.09995	4.99966
17	100.00000	100.00499	4.99980
18	100.00000	100.00025	4.99988

IEEE-754 'float' gives 1 decimal place, then explodes!

IEEE-754 'double' manages 2 decimal places, then explodes in the exact same way!

Essentially all floating-point systems converge on 100. It's easy to overlook the wrong answer.

What if we had *exact* arithmetic?

- Control the precision of the output
- Compute intermediate results to whatever level of precision is needed
- Replace 'number' with 'algorithm to compute the number'

Exact arithmetic package

Tcllib `math::exact` package. (Pure Tcl.)

Numbers (the result of calculations) are TclOO objects.

Reference counted objects – which is unpleasant.

Created by `math::exact::exactexpr`

Can format themselves with `asPrint` and `asFloat` methods.

Methods take desired precision. (Precision is the sum of the number of bits of exponent plus the number of bits of significand.)

Exact arithmetic - example

```
% package require math::exact
1.0
% namespace import math::exact::exactexpr
% set r [[exactexpr {exp(pi()*sqrt(163))}] ref]
::oo::Obj118
% $r asFloat 108
2.62537412640768e17
% $r asFloat 200
2.6253741264076874399999999999992500725971981e17
% set f [[exactexpr {$r-262537412640768744}] ref]
::oo::Obj125
% $f asFloat 100
-7.49927402801814311e-13
% $f unref
% $r unref
```

Exact arithmetic: fixing the quadratic formula

```
proc exactquad {a b c} {  
  set d [[exactexpr {sqrt($b*$b - 4*$a*$c)}] ref]  
  set r0 [[exactexpr {(-$b - $d)  
                    / (2 * $a)}] ref]  
  set r1 [[exactexpr {(-$b + $d)  
                    / (2 * $a)}] ref]  
  $d unref  
  return [list $r0 $r1]  
}
```

Exact arithmetic: fixing the quadratic formula

```
set a [[exactexpr 1] ref]
set b [[exactexpr 200] ref]
set c [[exactexpr {(-3/2) * 10**-12}] ref]
lassign [exactquad $a $b $c] r0 r1
$a unref; $b unref; $c unref
puts [list [$r0 asFloat 70] [$r1 asFloat 110]]
$r0 unref; $r1 unref
```

-2.000000000000000000000075e2 7.499999999999999999719e-15

No manipulation of the formulas! Just ask for 70 “bits” for the larger root and 110 “bits” for the smaller, and you get them.

Major disadvantages

- SLOW – but for generating constants in advance (for use in algorithms or for testing), probably ok
- No comparison operators (I'll explain...)
- No floating point notation on input
- Ref counting rather than automatic reclamation

No comparison?

- Comparison of reals isn't decidable.
- What you can have are operators

$$a <_{\epsilon} b, a >_{\epsilon} b, a =_{\epsilon} b, \dots$$

which are allowed to return the wrong answer if

$$|b - a| \leq \epsilon$$

These comparisons *are* computible.
Need a notation for specifying these in expressions.

No floating point notation?

What does `0.33333333333333333333` mean?

- The fraction $\frac{33333333333333333333}{100000000000000000000}$, exactly representing the string representation?
- The fraction $\frac{6004799503160661}{18014398509481984}$, exactly representing the internal representation?
- The fraction $\frac{1}{3}$, which will have string and internal representations as above?

For now, say what you mean! (Suggestions for what's reasonable are welcome!)

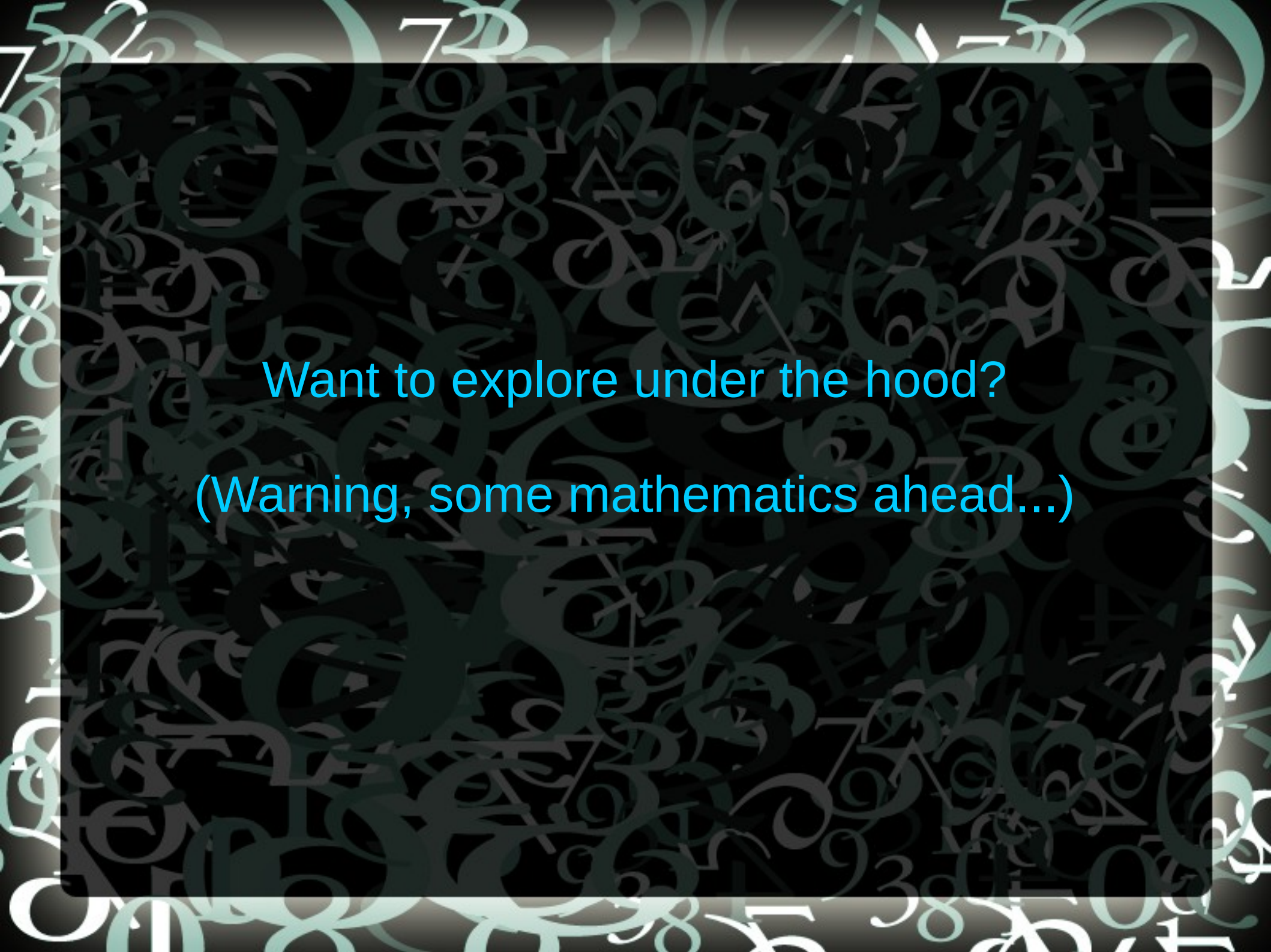
No garbage collection?

- The fragile reference problem
- Common to all interfaces between Tcl and managed code systems (e.g., Java, .NET)
- Bigger topic than just exact reals
- Generic extension for adding fragile references exists. (Implemented originally for TCOM)

<http://www.vex.net/~cthuang/counted/>



How am I doing for time?



Want to explore under the hood?
(Warning, some mathematics ahead...)

Digit streams

Digit streams, at first look attractive.

Make each algorithm into a coroutine that returns decimal or binary digits.

But there is a problem: things get jammed up!

What is $1/6 \times 9$ in decimal?

(It is 1.5, right?)

Recall that $1/6 = 0.16666\dots$

$$1/6 \times 9$$

$$0.9 \leq 0.1\dots \times 9 \leq 1.8$$

Not enough information to output a digit

$$1.44 \leq 0.16\dots \times 9 \leq 1.53$$

Output the leading digit 1

$$1.494 \leq 0.166\dots \times 9 \leq 1.503$$

$$1.4994 \leq 0.1666\dots \times 9 \leq 1.5003$$

$$1.49994 \leq 0.16666\dots \times 9 \leq 1.50003$$

I think we're stuck in a loop.

What about continued fractions?

$$x = a + \frac{1}{b + \frac{1}{c + \frac{1}{\ddots}}}$$

Terminate for all the rationals.

Periodic for quadratic surds.

The same problem with getting jammed – just more complicated expressions. What is $\text{sqrt}(2)**2$?

$$\text{Sqrt}(2) \leq 1 + 1/2 (1.5)$$

$$\text{Sqrt}(2) \geq 1 + 1/(2+1) (1.333\dots)$$

$$\text{Sqrt}(2) \geq 1 + 1/(2+1/2) (1.4)$$

$$\text{Sqrt}(2) \leq 1 + 1/(2+1/(2+1)) (1.42857\dots)$$

$$\text{Sqrt}(2) \leq 1 + 1/(2+1/(2+1/2)) (1.41666\dots)$$

$$\text{Sqrt}(2) \geq 1 + 1/(2+1/(2+1/(2+1))) (1.41176\dots)$$

$$\text{Sqrt}(2)**2 \leq 2.25$$

$$\text{Sqrt}(2)**2 \geq 1.777\dots$$

$$\text{Sqrt}(2)**2 \geq 1.96$$

$$\text{Sqrt}(2)**2 \leq 2.04\dots$$

$$\text{Sqrt}(2)**2 \leq 2.0069444\dots$$

$$\text{Sqrt}(2)**2 \geq 1.993\dots$$

We can never decide whether the a term of the result is 1 or 2!

Why do these ideas fail?



- If numbers are programs, exact comparison (equality, for example) is the Halting Problem.
- Alan Turing was originally an analyst, not a logician.
- Turing's work on the Halting Problem was a byproduct of his attempt to put the real numbers on a sound theoretical footing.

Exact comparison of real numbers is not decidable!

Use weak comparisons

$$a <_{\epsilon} b, a >_{\epsilon} b, a =_{\epsilon} b, \dots$$

Allowed to return the wrong answer if

$$|b - a| \leq \epsilon$$

Weak comparisons *are* computible.

Use redundant representations

More than one way to write a given number

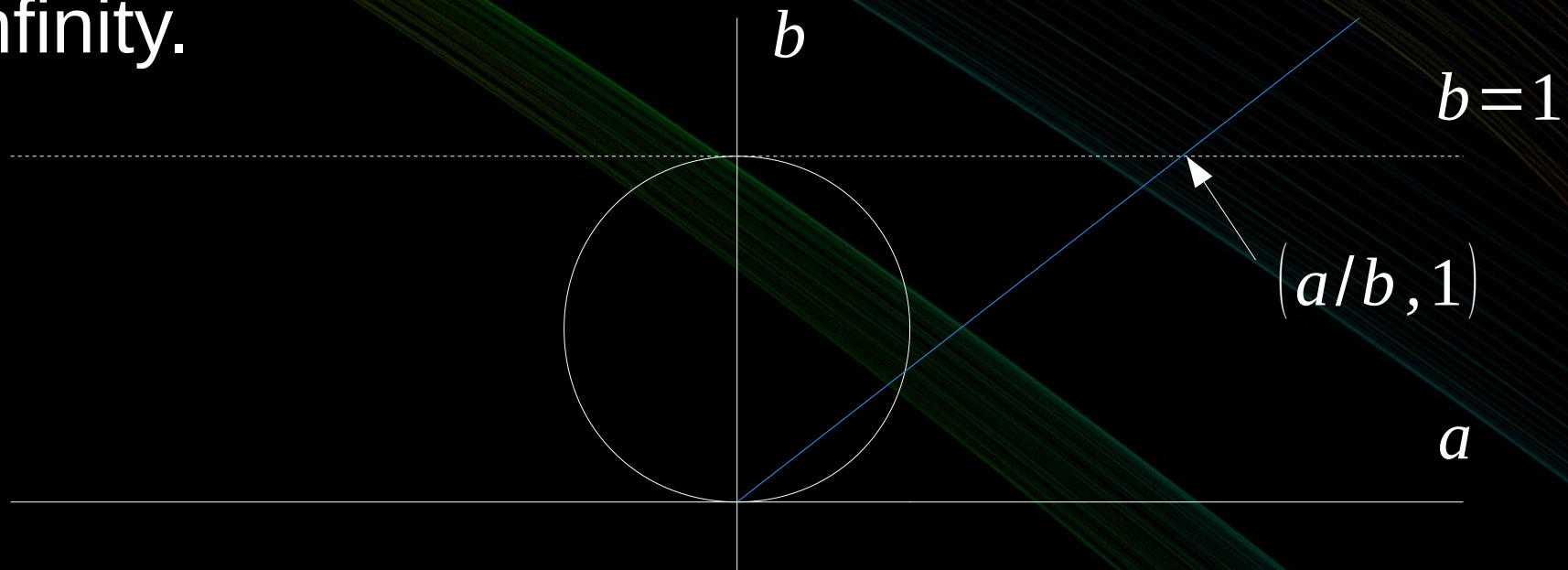
Can choose at least one correct way to write a number using only weak comparisons.

Rational numbers and integer vectors

Represent the rational number a/b by the vector $\{a \ b\}$

(Assume reduction to lowest terms).

Division by zero is OK, and there is a single point at infinity.



Möbius transformations

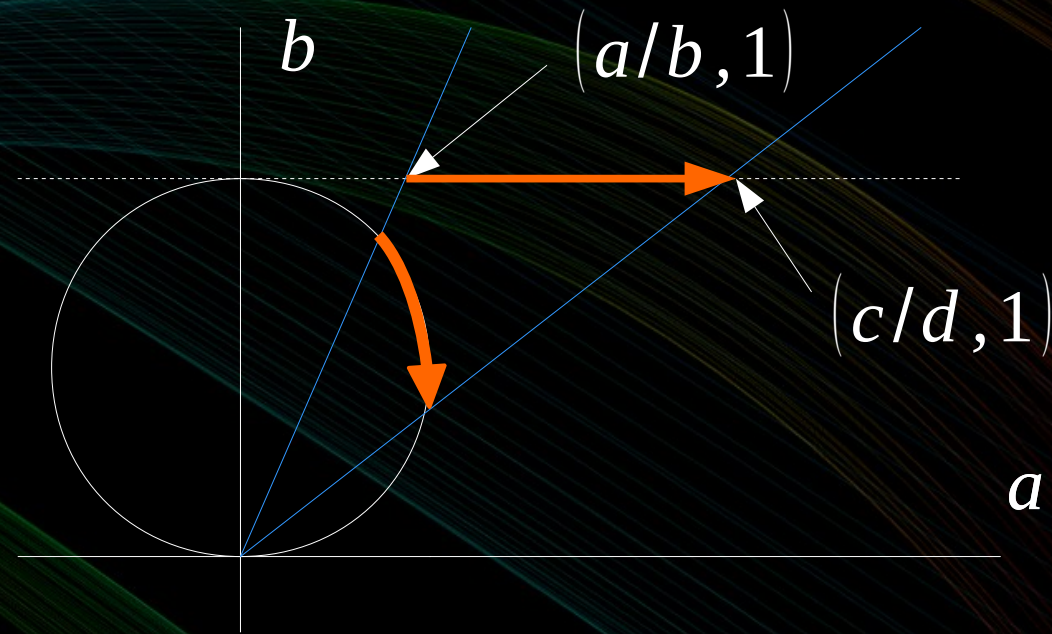
The matrix

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

represents the function

$$\frac{ax+c}{bx+d}$$

If $x > 0$, this is also the generalized interval $[c/d .. a/b]$, moving clockwise on the circle



Composition of transformations

$$\text{If } f(x) = \frac{ax+c}{bx+d}, \text{ and } g(x) = \frac{ex+g}{fx+h},$$
$$\text{then } f(g(x)) = \frac{(ae+cf)x+(ag+ch)}{(be+df)x+(bg+dh)}$$

This is just matrix multiplication:

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} \begin{bmatrix} e & g \\ f & h \end{bmatrix} = \begin{bmatrix} ae+cf & ag+ch \\ be+df & bg+dh \end{bmatrix}$$

Relation to continued fractions

$$a + \frac{1}{b + \frac{1}{c + \frac{1}{\ddots}}} \text{ can be written } \begin{bmatrix} a & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} b & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} c & 1 \\ 1 & 0 \end{bmatrix} \cdot \dots$$

Real numbers are streams of matrices.

Bilinear transformations

We also use $2 \times 2 \times 2$ tensors. If x and y are vectors, write:

$$\left[\begin{array}{cc|cc} a & e & c & g \\ b & f & d & h \end{array} \right] L x R y, \text{ meaning } \frac{a x y + b x + c y + d}{e x y + f x + g y + h}$$

If x and y are vectors, A and B are matrices, \mathcal{T} is a tensor, all of the following are well defined:

$A \cdot x$	vector	$T L x$	matrix	$T R y$	vector
$A \cdot B$	matrix	$T L A$	matrix	$T R B$	matrix
$A \cdot \mathcal{T}$	tensor				

Arithmetic on exact reals

Remembering that

$$\left[\begin{array}{cc|cc} a & e & c & g \\ b & f & d & h \end{array} \right] L \times R y \text{ means } \frac{a x y + b x + c y + d}{e x y + f x + g y + h}$$

The following special tensors will be useful.

$$\begin{array}{cc} \left[\begin{array}{cc|cc} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] & \text{addition} & \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] & \text{multiplication} \\ \left[\begin{array}{cc|cc} 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] & \text{subtraction} & \left[\begin{array}{cc|cc} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right] & \text{division} \end{array}$$

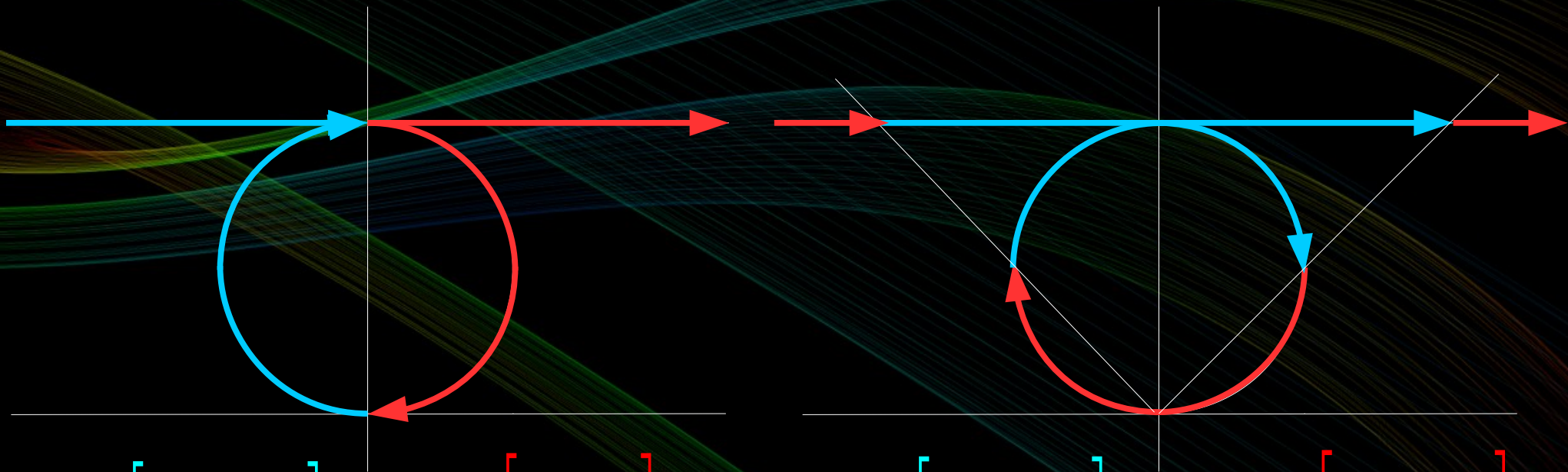
Expressions are trees of tensors, matrices, vectors

- Special functions and non-integer powers are infinite trees
- Lazy evaluation. A node in a special function knows how to instantiate its children and does so only when it needs to.
- Example:

$$\sqrt{x} = \begin{bmatrix} x+1 & 2x \\ 2 & x+1 \end{bmatrix} \cdot \begin{bmatrix} x+1 & 2x \\ 2 & x+1 \end{bmatrix} \cdot \begin{bmatrix} x+1 & 2x \\ 2 & x+1 \end{bmatrix} \cdots$$
$$\ln x = \begin{bmatrix} x-1 & x-1 \\ 1 & x \end{bmatrix} \cdot \prod_{n=1}^{\infty} \begin{bmatrix} nx+n+1 & (2n+1)x \\ 2n+1 & (n+1)x+n \end{bmatrix}$$

Sign matrices

A sign matrix selects half the number circle.



$$S_- = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

$$S_+ = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$S_0 = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

$$S_\infty = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

Every number's stream begins with a sign matrix.

Always have two choices – decidable.

Remaining matrices in the stream will have all elements nonnegative.

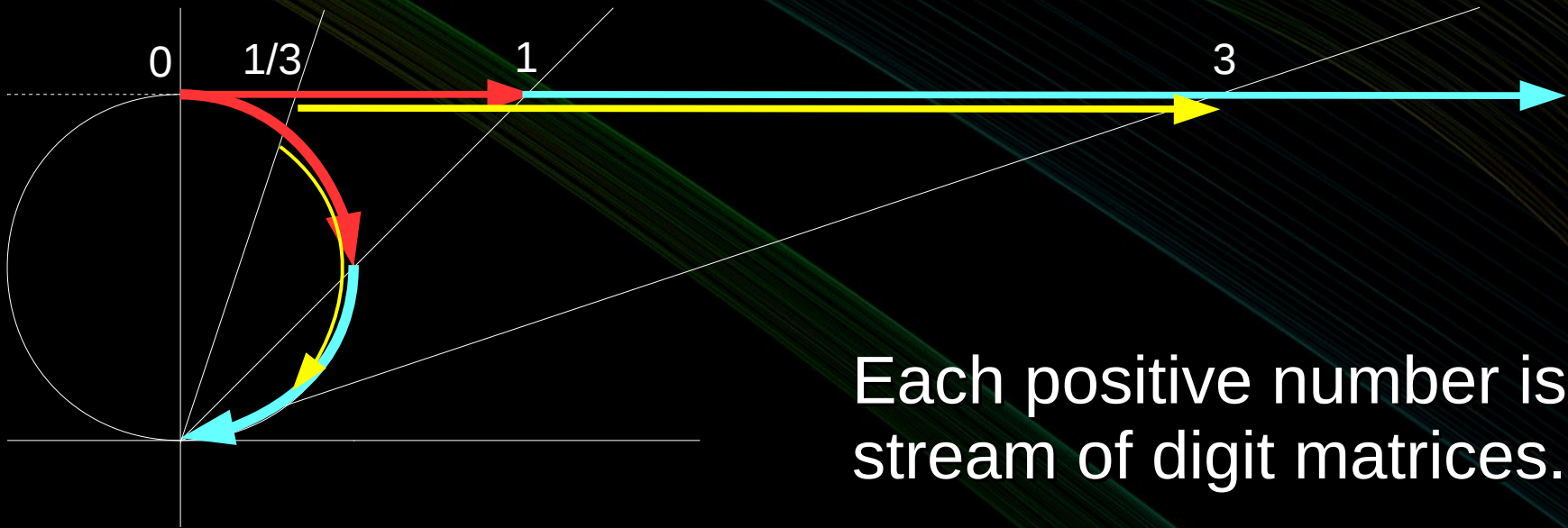
Digit matrices

- Digit matrices slice up the nonnegative half-line

$$D_- = \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix}$$

$$D_0 = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

$$D_+ = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}$$



Each positive number is a stream of digit matrices.

Each digit matrix conveys ≈ 1 bit of information.

Pulling the stream

- For each possible digit matrix D , a matrix M or tensor \mathcal{T} asks whether it can prove that $D^{-1}\chi$ is still positive.
- If so, it emits D and replaces itself with $D^{-1}\chi$.
- If no digit is emitted, then a matrix M asks for a digit E from its argument, multiplies itself by the digit and tries again.
- A tensor \mathcal{T} will ask its left and right arguments in succession and replace itself with $\mathcal{T}\mathbf{L}E$ or $\mathcal{T}\mathbf{R}E$ respectively
- Every digit can be delivered in a finite number of steps