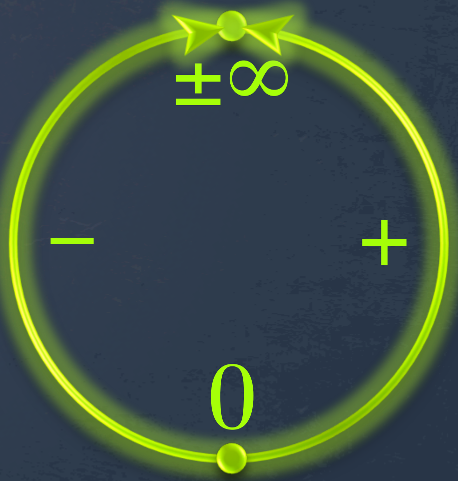
A Radical Approach to Computation with Real Numbers

John Gustafson
A*CRC and NUS



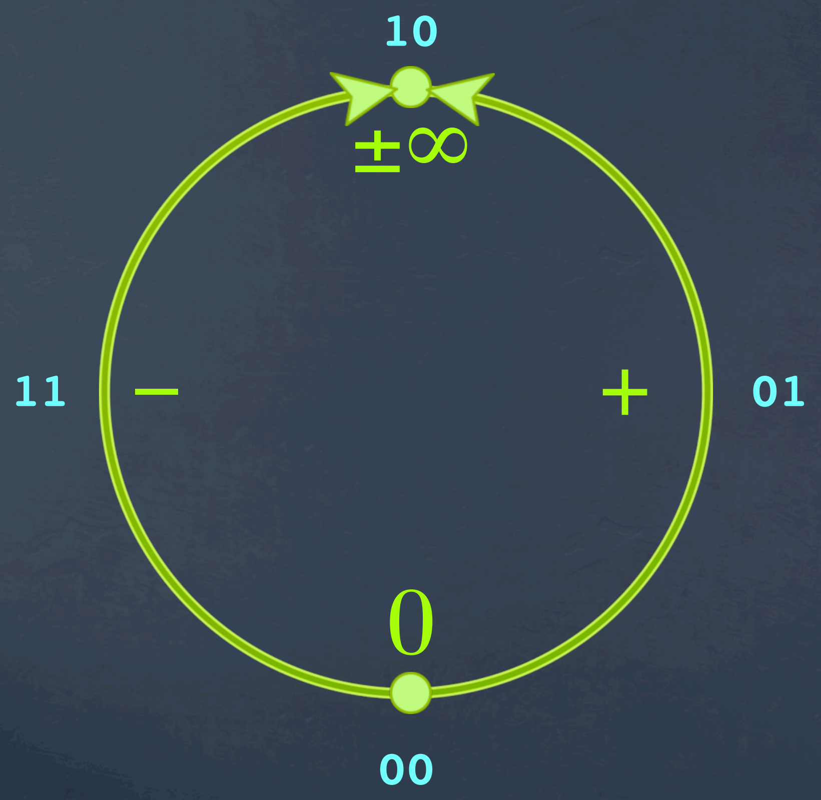
"Unums version 2.0"

Break *completely* from IEEE 754 floats and gain:

- Computation with mathematical rigor
- Robust set representations with a *fixed* number of bits
- 1-clock binary ops with *no* exception cases
- Tractable “exhaustive search” in high dimensions

Strategy: Get ultra-low precision right, **then** work up.

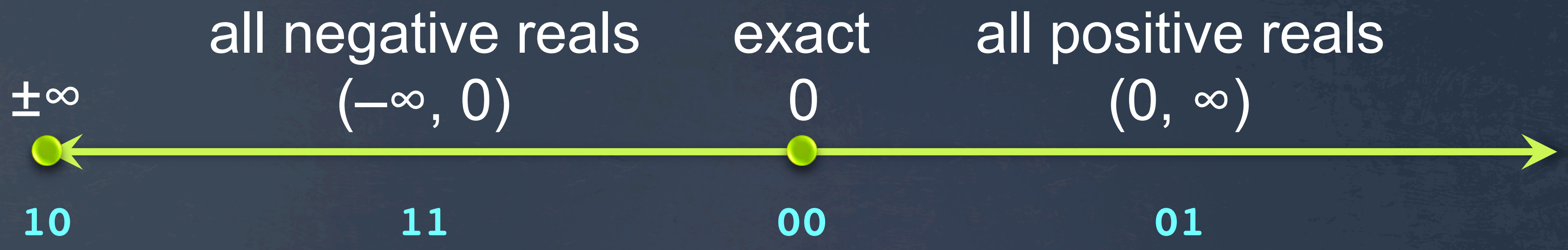
All projective reals, using 2 bits



"±∞" is "the point at infinity" and is *unsigned.*

Think of it as the reciprocal of zero.

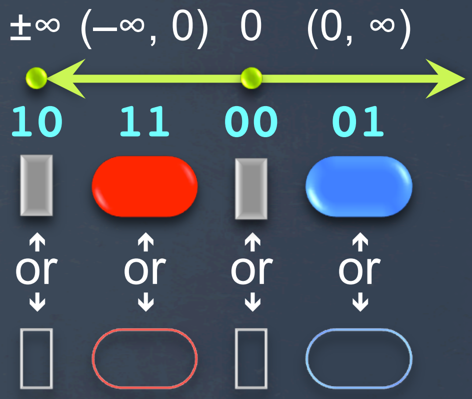
Linear depiction



Maps to the way 2s complement integers work!

Redundant point at infinity on the right is not shown.

Absence-Presence Bits



Forms the ***power set*** of the four states.
$2^4 = 16$ possible subsets of the extended reals.

0 (open shape) if absent from the set,
1 (filled shape) if present in the set.

Rectangle if exact, oval or circle if inexact (range)

Red if negative, blue if positive

Sets become *numeric quantities*

“SORNs”: Sets Of Real Numbers

▯	○	▯	○	The empty set, { }
▯	○	▯	●	All positive reals (0, ∞)
▯	○	▮	○	Zero, 0
▯	○	▮	●	All nonnegative reals, [0, ∞)
▯	●	▯	○	All negative reals, (–∞, 0)
▯	●	▯	●	All nonzero reals, (–∞, 0) ∪ (0, ∞)
▯	●	▮	○	All nonpositive reals, (–∞, 0]
▯	●	▮	●	All reals, (–∞, ∞)
▮	○	▯	○	The point at infinity, ±∞
▮	○	▯	●	The extended positive reals, (0, ∞]
▮	○	▮	○	The unsigned values, 0 ∪ ±∞
▮	○	▮	●	The extended nonnegative reals, [0, ∞]
▮	●	▯	○	The extended negative reals, [–∞, 0)
▮	●	▯	●	All nonzero extended reals [–∞, 0) ∪ (0, ∞]
▮	●	▮	○	The extended nonpositive reals, [–∞, 0]
▮	●	▮	●	All extended reals, [–∞, ∞]

Closed under

$x + y$ $x - y$

$x \times y$ $x \div y$

and… x^y

Tolerates division by 0.
No indeterminate forms.

Very different from *symbolic* ways of dealing with sets.

No more “Not a Number”

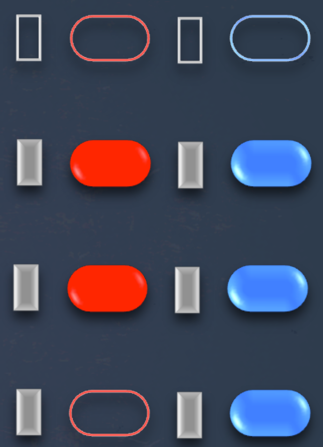
$\sqrt{-1}$ = empty set:

0 / 0 = everything:

∞ – ∞ = everything:

1^{∞} = all nonnegatives, [0, ∞]:

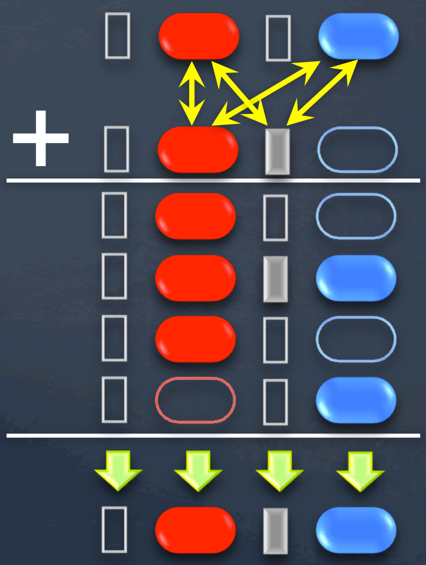
etc.



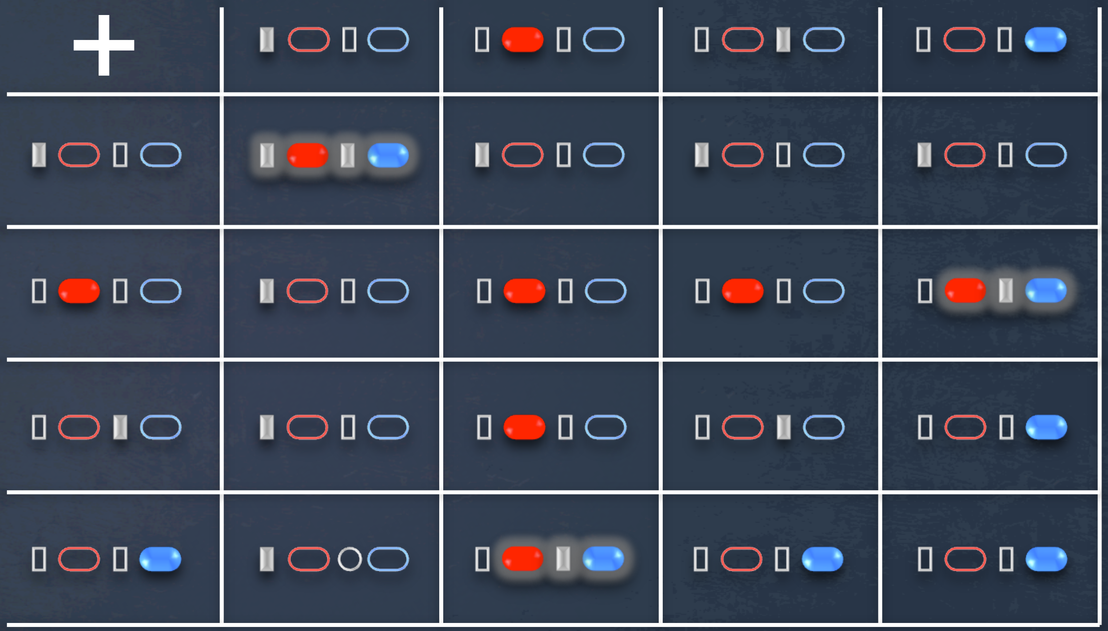
Answers, as limit forms, are *sets*. We can express those!

Op tables need only be 4x4

For any SORN, do table look-up for pairwise bits that are set, and find the union with a bitwise OR.

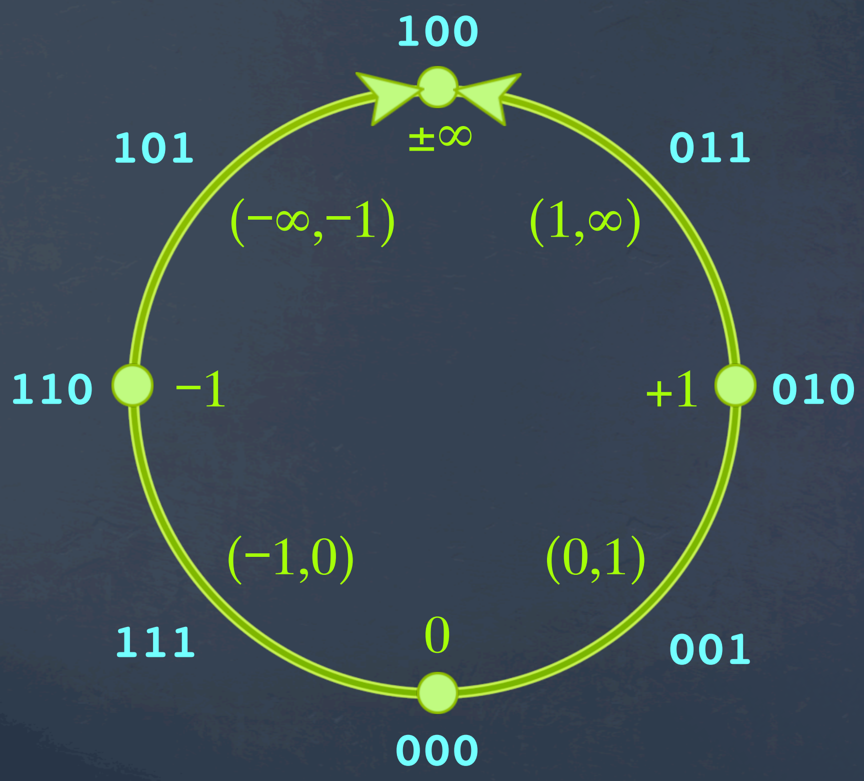


parallel OR



Note that three entries “blur”, indicating *information loss.*

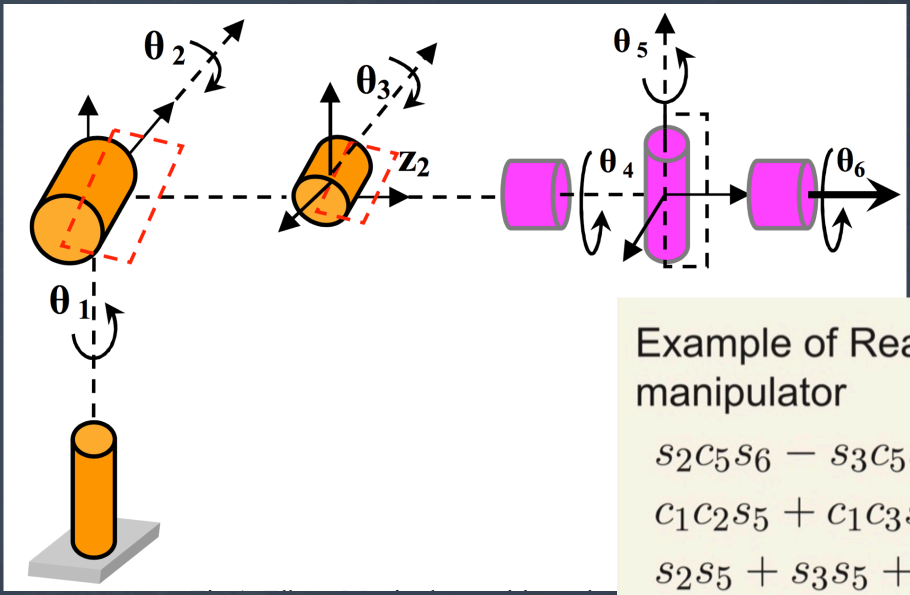
Now include +1 and –1



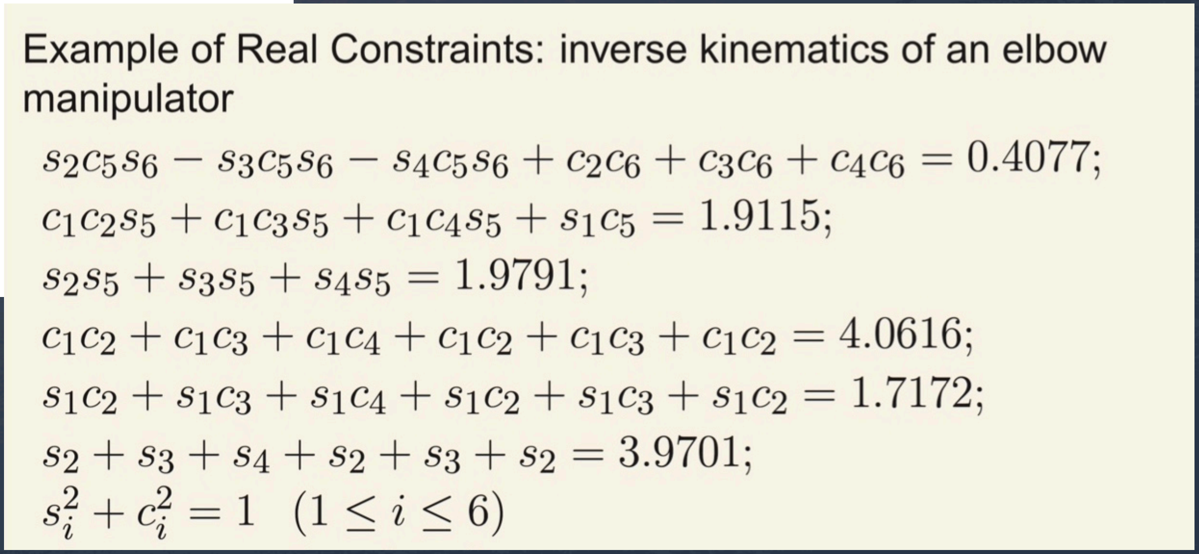
The SORN is 8 bits long.

This is actually enough of a number system to be useful!

Example: Robotic Arm Kinematics



12-dimensional nonlinear system (!)



Example of Real Constraints: inverse kinematics of an elbow manipulator

$$s_2c_5s_6 - s_3c_5s_6 - s_4c_5s_6 + c_2c_6 + c_3c_6 + c_4c_6 = 0.4077;$$
$$c_1c_2s_5 + c_1c_3s_5 + c_1c_4s_5 + s_1c_5 = 1.9115;$$
$$s_2s_5 + s_3s_5 + s_4s_5 = 1.9791;$$
$$c_1c_2 + c_1c_3 + c_1c_4 + c_1c_2 + c_1c_3 + c_1c_2 = 4.0616;$$
$$s_1c_2 + s_1c_3 + s_1c_4 + s_1c_2 + s_1c_3 + s_1c_2 = 1.7172;$$
$$s_2 + s_3 + s_4 + s_2 + s_3 + s_2 = 3.9701;$$
$$s_i^2 + c_i^2 = 1 \quad (1 \leq i \leq 6)$$

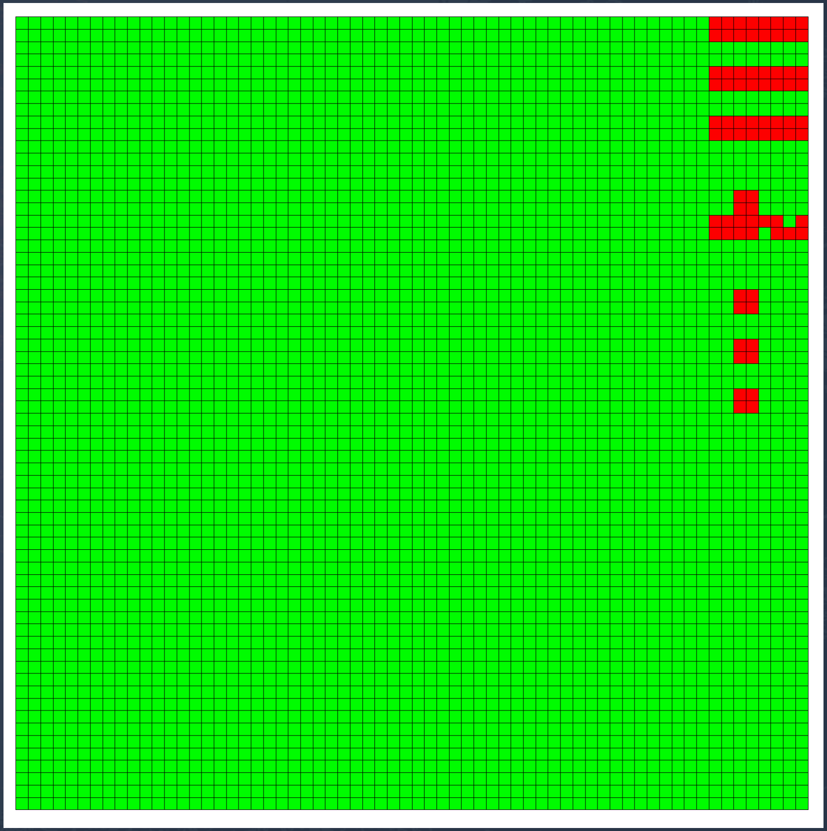
Notice all values must be in [–1,1] ➔

“Try everything”… in 12 dimensions

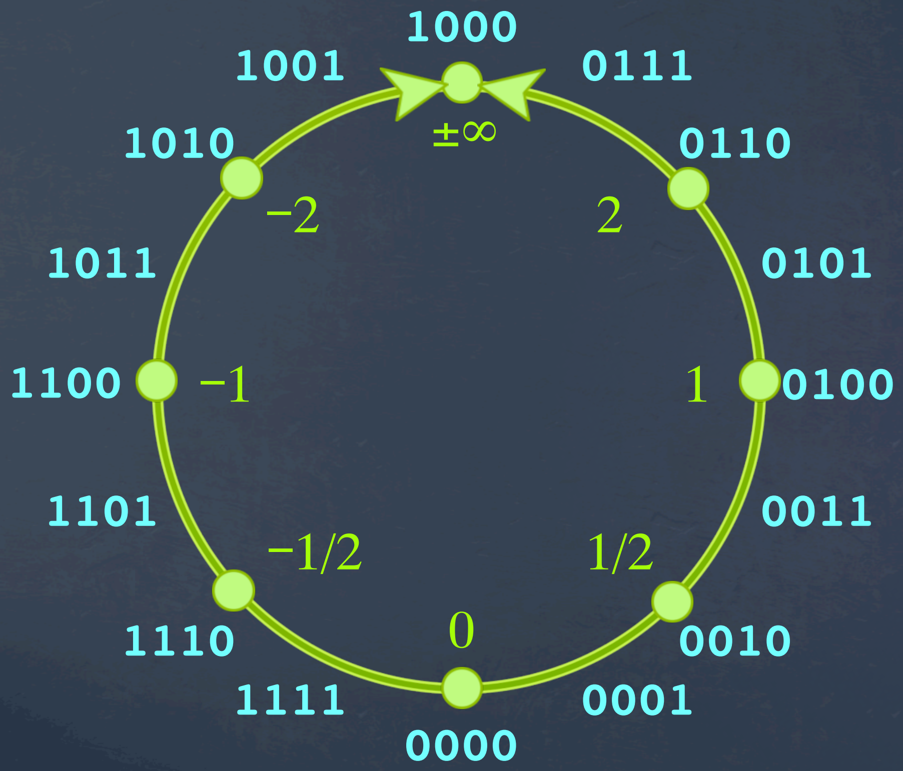
Every variable is in [-1,1], so split into [-1,0) and [0,1] and compute the constraint function to 3-bit accuracy.

🟩= violates constraints
🟥= compliant subset

2^{12} = 4096 sub-cubes can be evaluated in parallel, in a few *nanoseconds*.

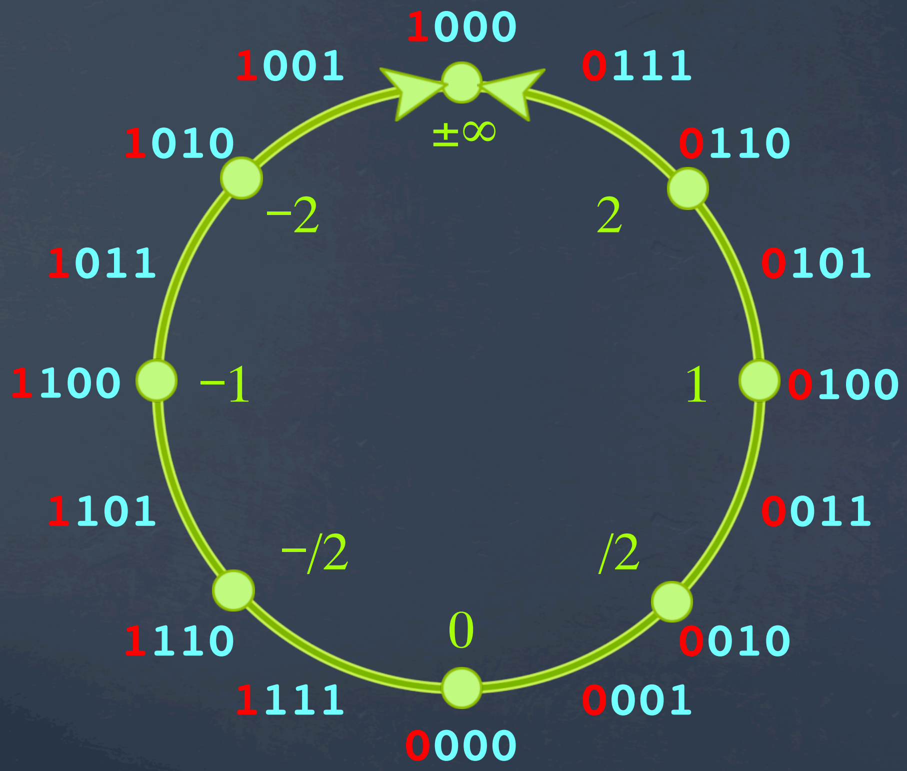


One option: more powers of 2



There is nothing special about 2. We could have added 10 and 1/10, or even *π* and 1/*π*, or *any exact number*.
(Yes, *π* can be numerically exact, if we want it to be!)

Note: sign bit is in the usual place

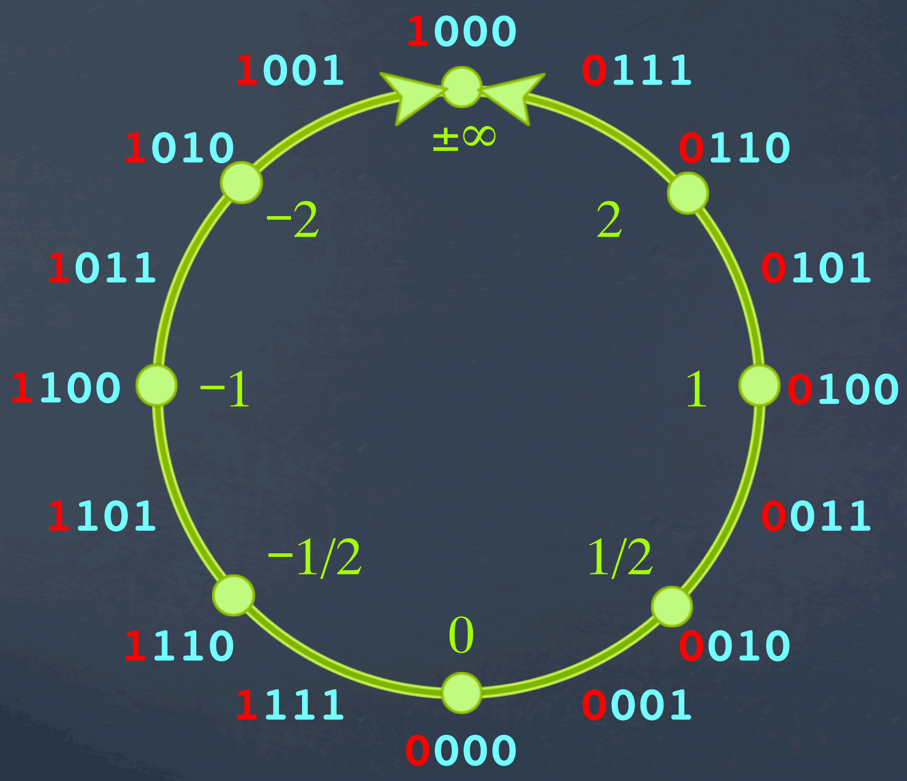


The sign of 0 and ±∞ is meaningless, since

0 = –0 and

±∞ = –±∞.

Negation is trivial



To negate, flip horizontally.

Reminder: In 2’s complement, flip all bits and add **1**, to negate. *Works without exception, even for* 0 *and* ±∞. (They do not change.)

A new notation: Unary “/”

Just as unary “–” can be put before x to mean $0 - x$,
unary “/” can be put before x to mean $1/x$.

Just as we can write $-x$ for $0 - x$, we can write $/x$ for $1/x$. Pronounce it “over x”

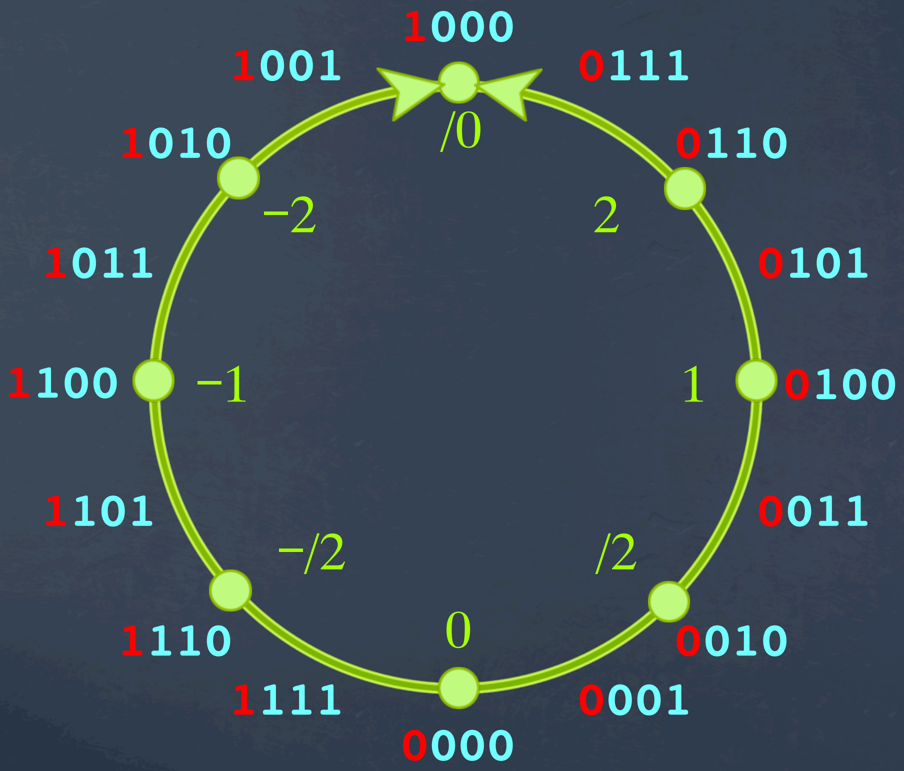
Parsing is just like parsing unary minus signs.

$-(-x) = x$, just as $/(/x) = x$.
$x - y = x + (-y)$, just as $x \div y = x \times (/y)$

These unum number systems are always lossless
(no rounding error) under negation ***and*** reciprocation.

Arithmetic ops $+\ -\ \times\ \div$ are finally put on **equal footing**.

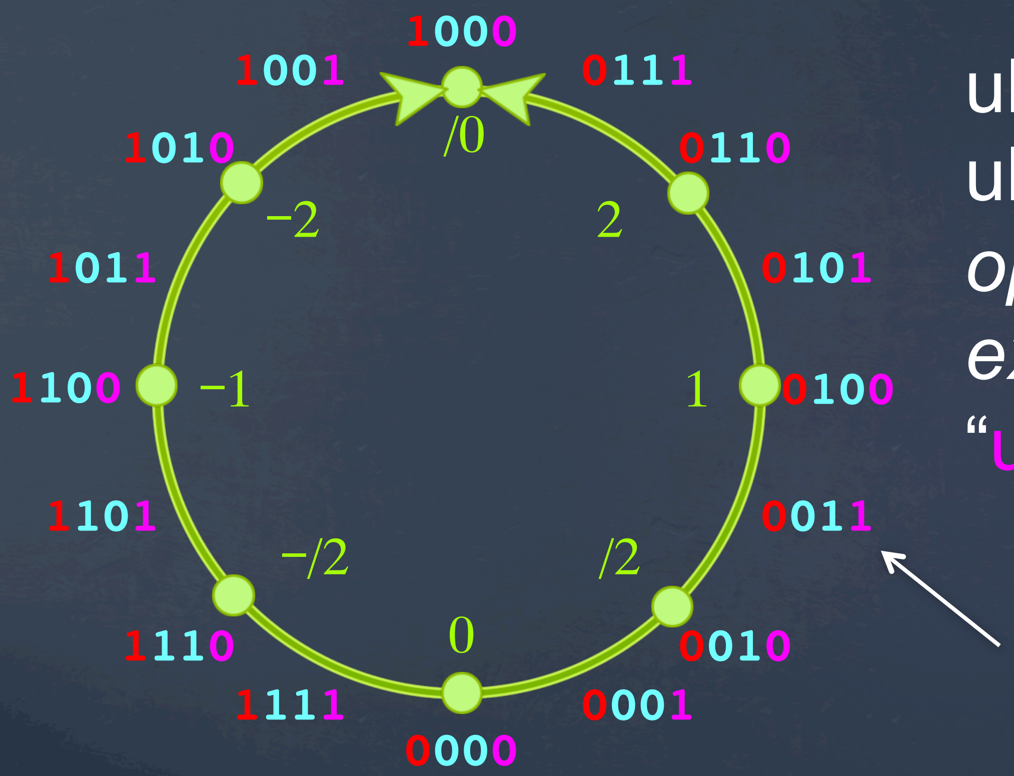
Reciprocation is trivial, too!



To reciprocate, flip *vertically*.

Reverse all bits but the first one and add **1**, to reciprocate. *Works without exception*. +1 and –1 do not change.

The last bit serves as the *ubit*



ubit = **0** means exact
ubit = **1** means *the open interval between exact numbers*.
"uncertainty bit".

Example: This means the open interval (½, 1). Or (get used to it), (/2, 1).

Back to kinematics, with exact 2^k

Split one dimension at a time. Needs only 1600 function evaluations (microseconds).

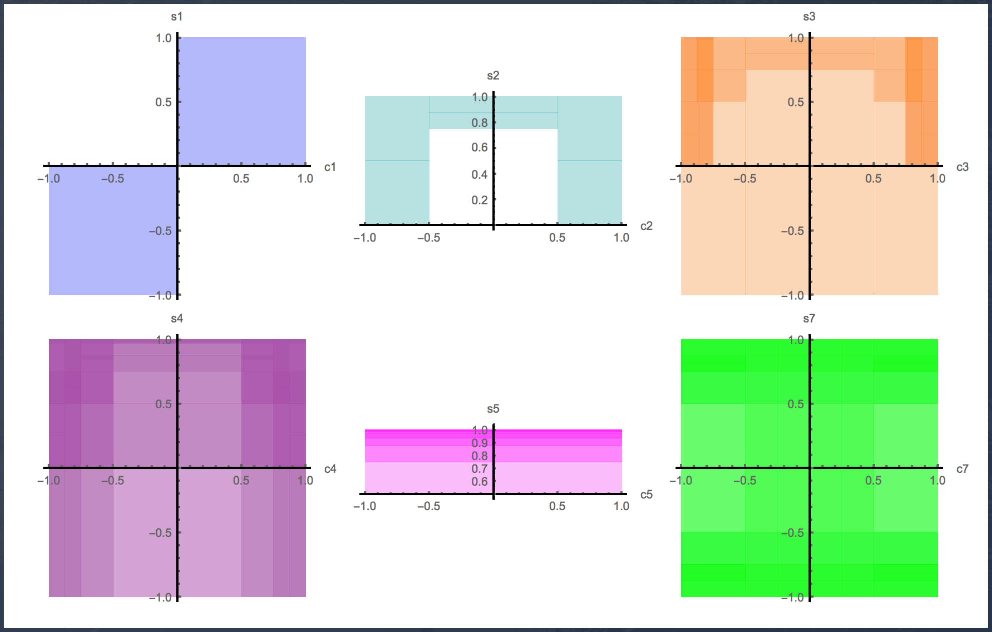
Display six 2D graphs of *c* versus *s* (cosine versus sine… should converge to an arc)

Here is what the *rigorous bound* looks like after one pass.

Information = /uncertainty.

Uncertainty = answer volume.

Information increases by 1661×



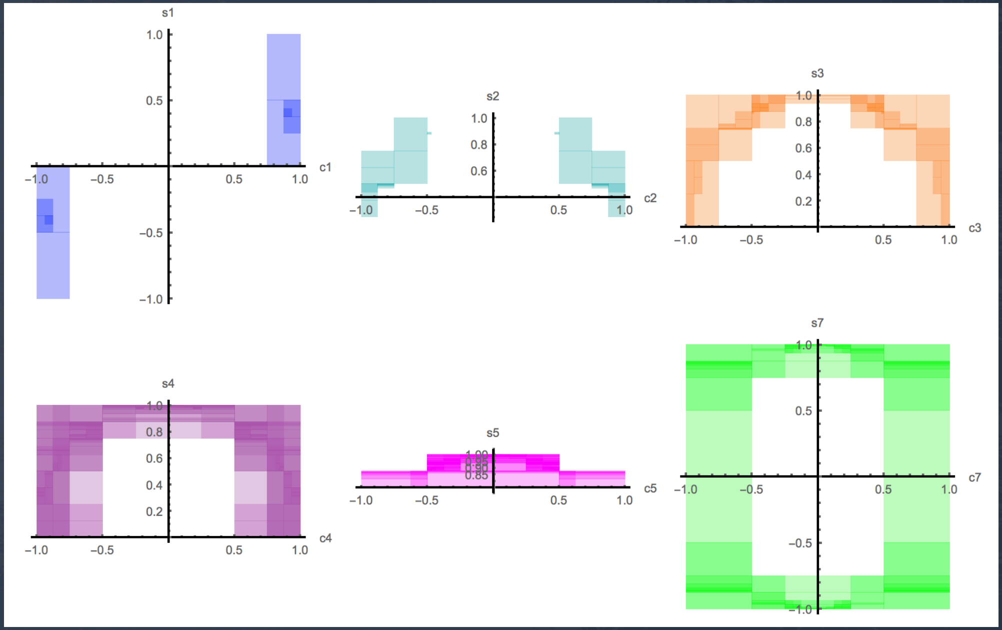
Make a second pass

Still using ultra-low precision

Starting to look like arcs (angle ranges)

457306 function evaluations (milliseconds if no parallelism used)

Information increases by a factor of 3.7×10^6



A third pass allows robot decision

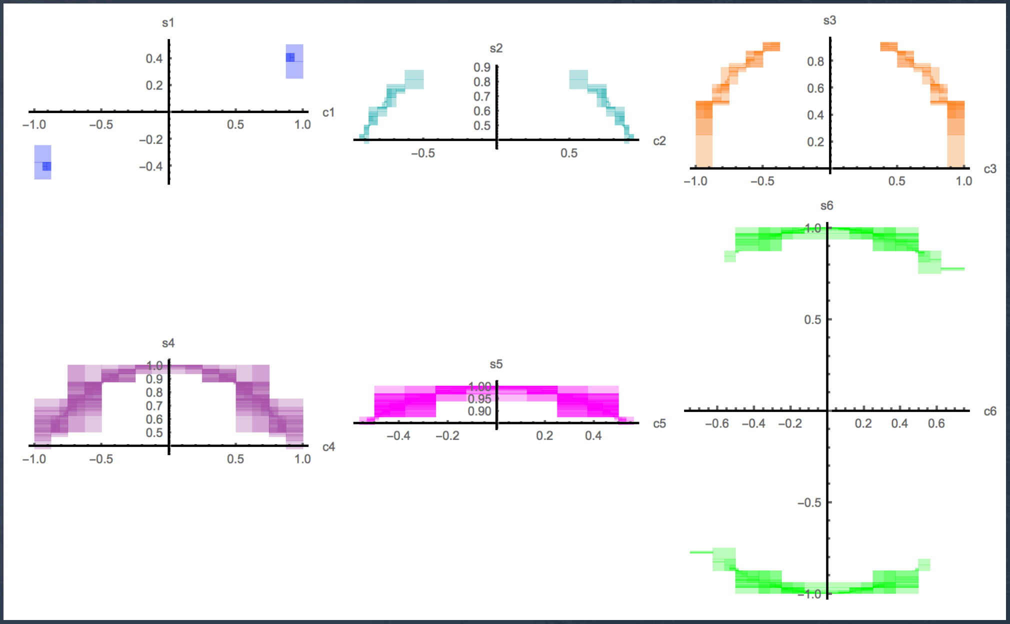
Transparency helps display 12 dimensions, 2 at a time.

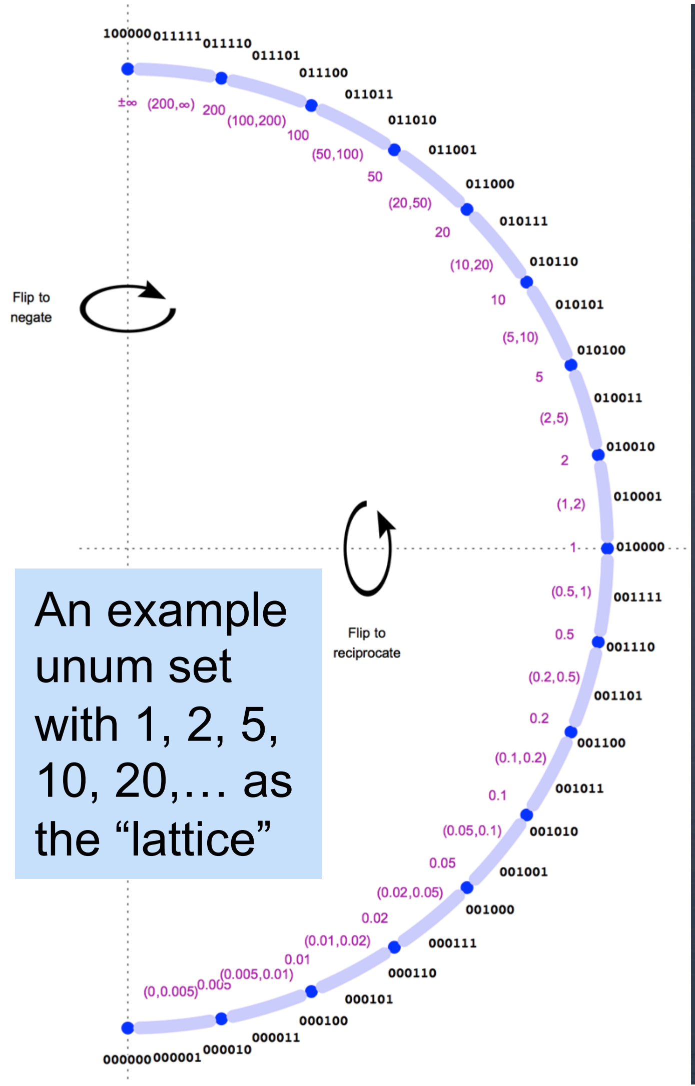
Starting to look like arcs (angle ranges).

6 million function evaluations (milliseconds, with parallelism)

Information increases by a factor of 1.8×10^{11}

Remember, this is a **rigorous bound** of all possible solutions. Gradient-type searching with floats can only ***guess***.





An example unum set with 1, 2, 5, 10, 20,… as the “lattice”

Unums II

Universal Numbers. They are like the original unums, but:

- Fixed size
- *Not* an extension of IEEE floats
- ULP size variance becomes *sets*
- No redundant representations
- No wasted bit patterns
- No NaN exceptions
- No penalty for using decimals!
- No errors in converting human-readable format to and from machine-readable format.

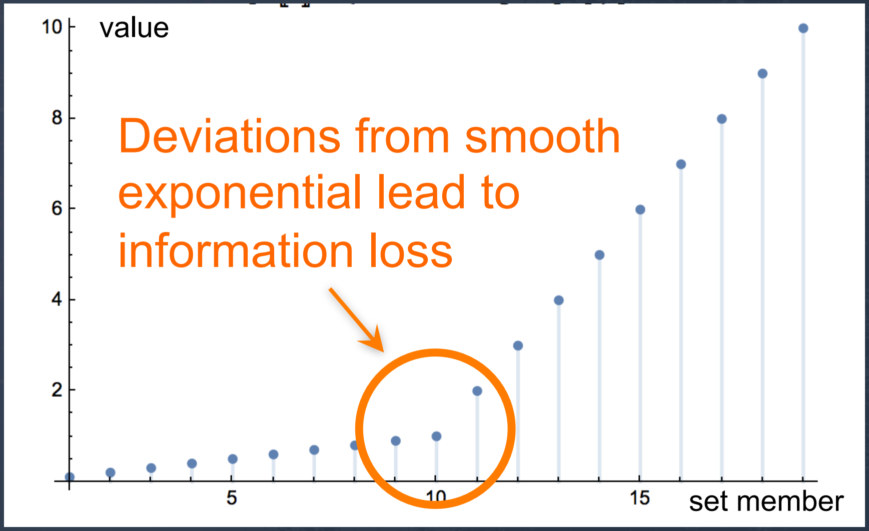
Time to get serious

What is the best possible use of an *8-bit byte* for real-valued calculations?

Start with kindergarten numbers:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Divide by 10 to center the set about 1:
0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
1, 2, 3, 4, 5, 6, 7, 8, 9, 10

This has the classic problem with decimal IEEE floats: "***wobbling precision***."



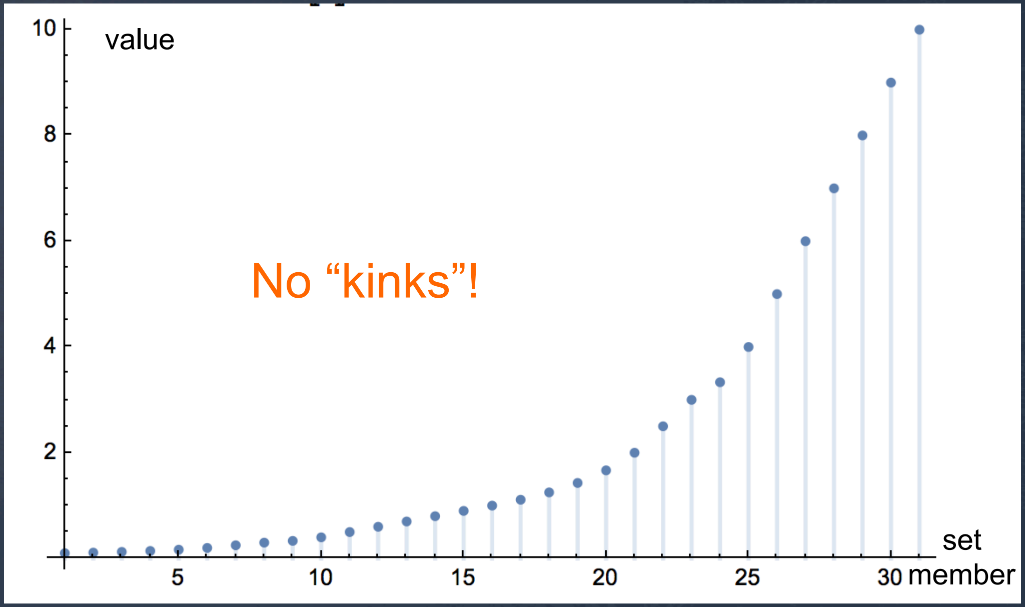
Reciprocal closure cures wobbling precision

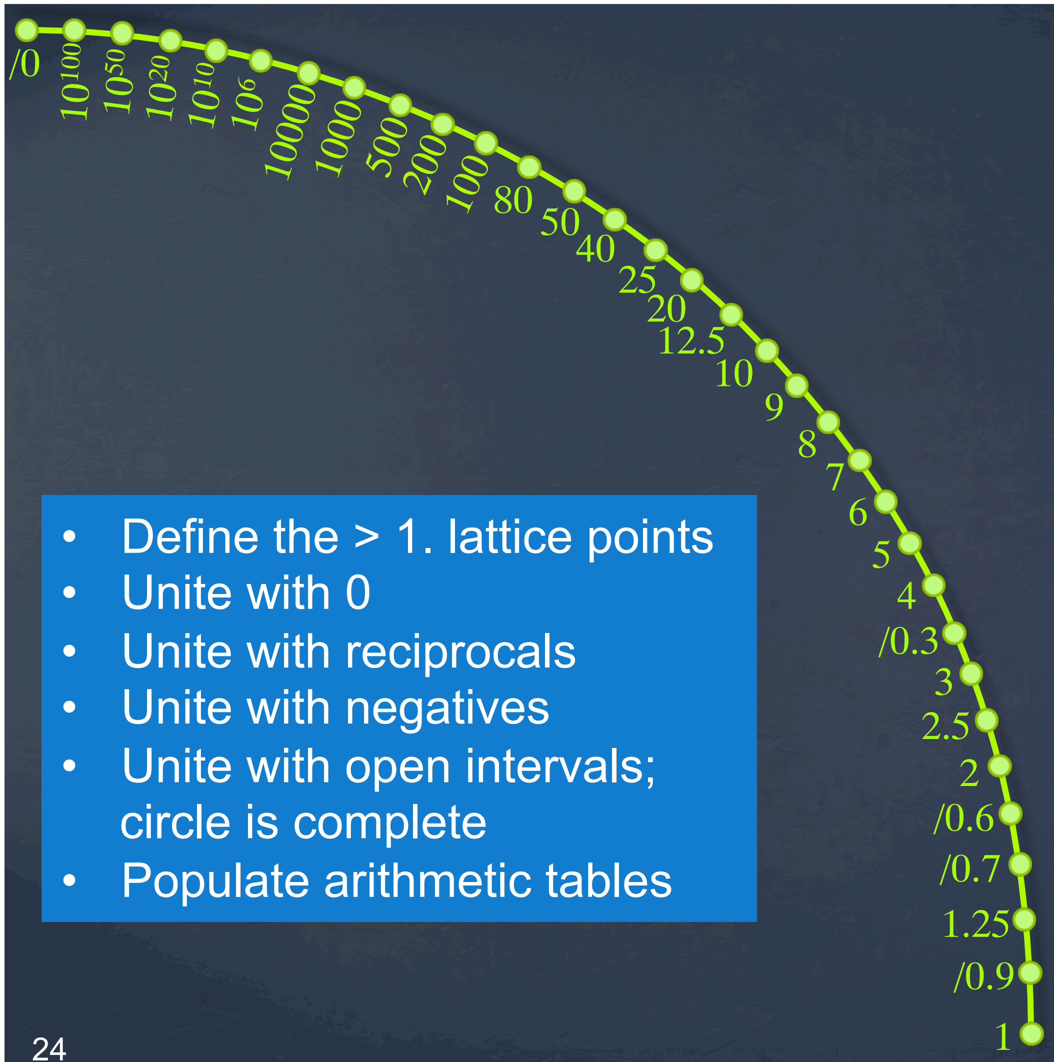
Unite set with the reciprocals of the values, guaranteeing closure:

0.1, /9, 0.125, /7, /6, 0.2, 0.25, 0.3, /3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,

1, /0.9, 1.25, /0.7, /0.6, 2, 2.5, 3, /0.3, 4, 5, 6, 7, 8, 9, 10

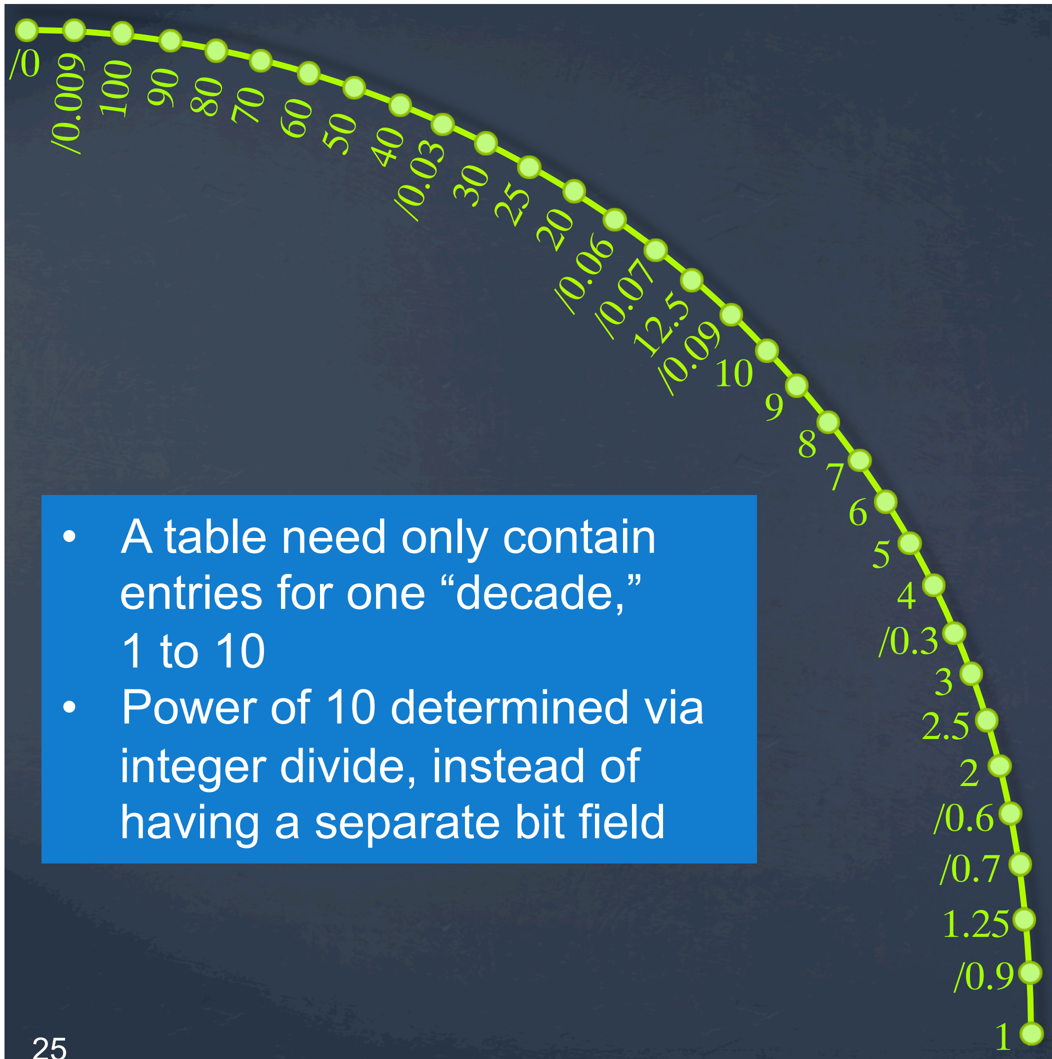
That's 30 numbers. Room for 33 more.





- Define the > 1. lattice points
- Unite with 0
- Unite with reciprocals
- Unite with negatives
- Unite with open intervals; circle is complete
- Populate arithmetic tables

"Tapered Precision" reduces relative accuracy for extreme magnitudes, allowing larger dynamic range.

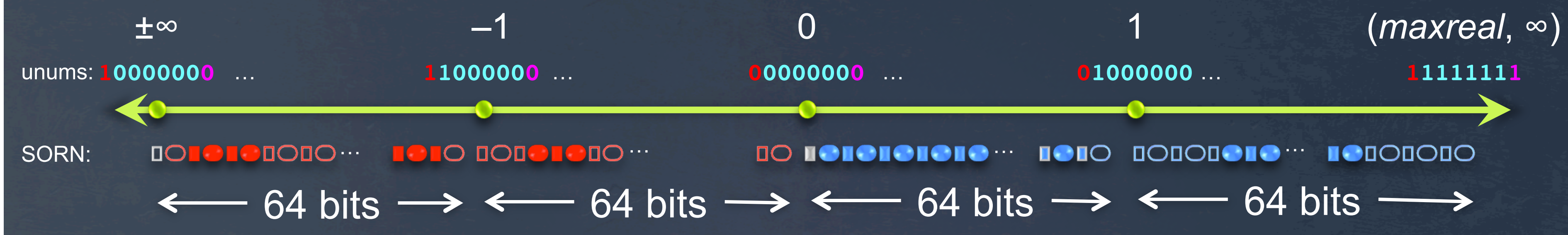


- A table need only contain entries for one “decade,” 1 to 10
- Power of 10 determined via integer divide, instead of having a separate bit field

Flat precision makes table generation and fused operations easier.

Imagine: custom number systems for *application-specific arithmetic*

8-bit unum means 256-bit SORN



Ultra-fast parallel arithmetic on *arbitrary* subsets of the real number line. Ops can still finish within a single clock cycle, with a tractable number of parallel OR gates.

16-bit SORN for + – × ÷ ops

Connected sets *remain connected* under + – × ÷, even division by zero!

Run-length encoding of a block of 1s amongst 256 bits only takes 16 bits.

```
00000000 00000000 means all 256 bits are 0s
11111111 11111111 means all 256 bits are 1s
00000010 00000110 means there is a block of 2 1s starting at position 6
      ↑        ↑
      2        6
```

Trivial logic still serves to negate and reciprocate compressed form of value.

Table look-up background

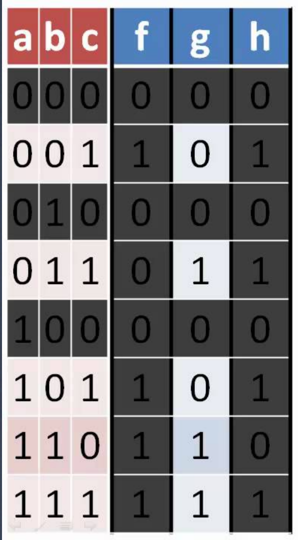
In 1959, IBM introduced its 1620 Model 1 computer, internal nickname "CADET".

All math was by table look-up.

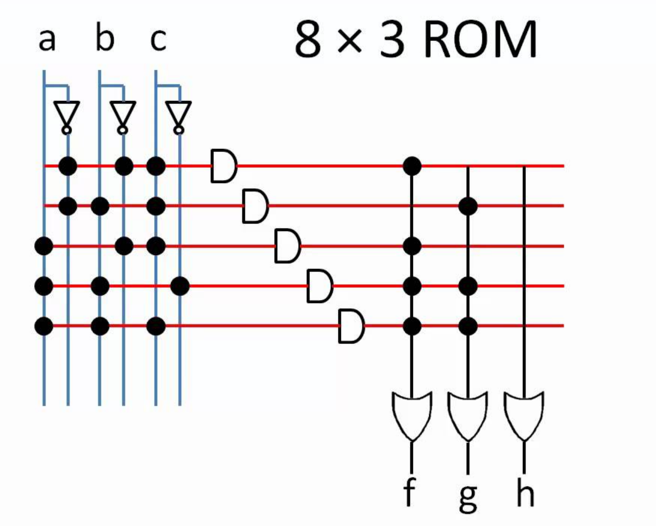
Customers decided CADET meant "Can't Add, Doesn't Even Try."

Table look-up requires ROM

- Read-Only Memory needs very few *transistors*.
- Billions of bits per chip, easy
- Imagine the *speed*… all operations take 1 clock! Even x^y.
- 1-op-per clock architectures are much easier to build, less silicon
- Single argument-operations require tiny tables. Trig, exp, you name it.



a	b	c	f	g	h
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	1



Low-precision *rigorous* math is possible at 100x the speed of sloppy IEEE floats.

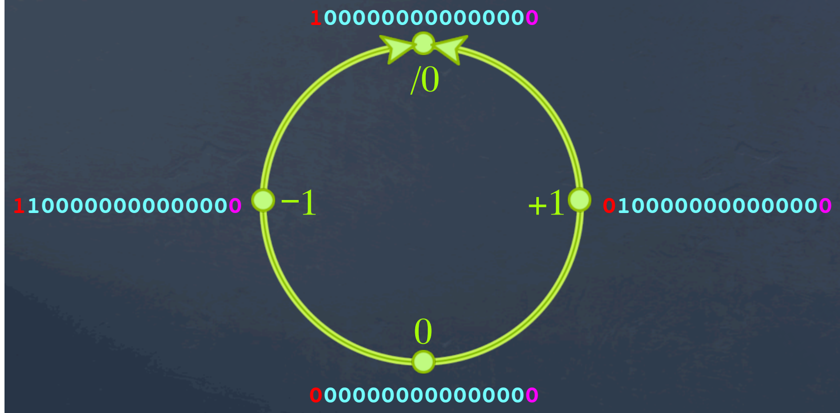
Cost of + – × ÷ tables

- Addition table: 256×256 entries, 2-byte entries = 128 kbytes
- Symmetry cuts that in half, if we sort *x* and *y* inputs so $x \leq y$
- Subtraction table: just use negative of addition table
- Multiplication table: same size as addition table
- Division table: just use reciprocal of multiplication table!
- Estimated chip cost: < 0.01 mm^2, < 1 milliwatts

128 kbytes total for all four basic ops.
Another 128 kbytes if we also table x^y.

What about, you know, *decent* precision? Is 3 decimals enough?

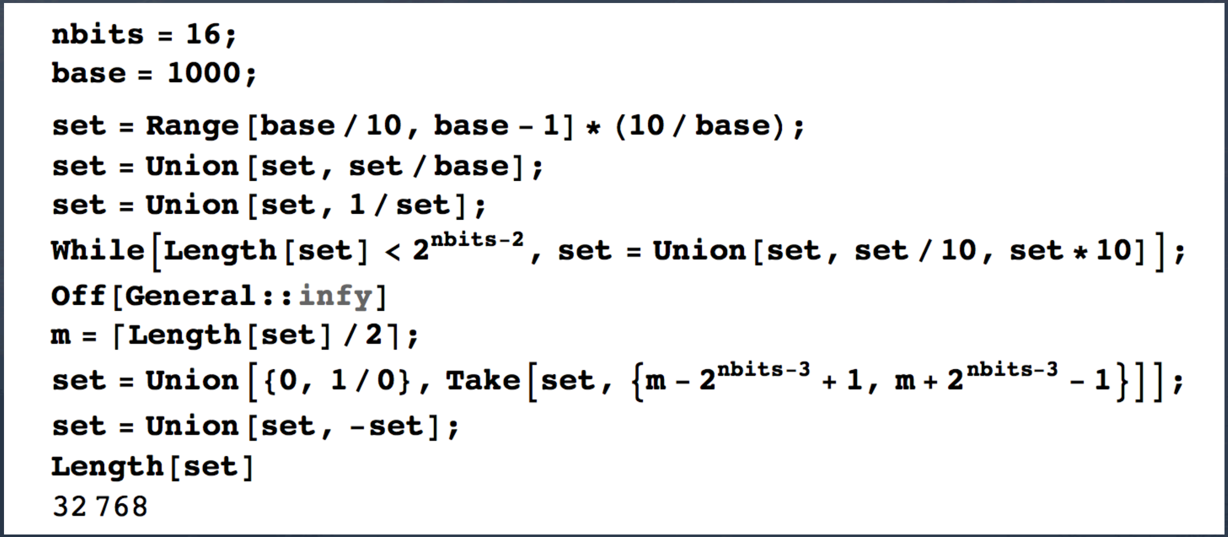
IEEE half-precision (16 bits) has ~3 decimal accuracy
9 orders of magnitude, 6×10^{-5} to 6×10^{4}.
Many bit patterns wasted on NaN, negative zero, etc.
Can a 16-bit unum do better, and *actually express decimals exactly*?



65536 bit patterns. 8192 in the "lattice".
Start with set = {1.00,1.01, 1.02,…, 9.99}.
Unite with reciprocals.
While set size < 16384: unite with 10× set.
Clip set to 16384 elements centered at 1.00
Unite with negatives.
Unite with open intervals between exacts.
What is the *dynamic range*?

Answer: ***10*** orders of magnitude

~8.7×10^{-6} to ~1.1×10^{5}



```
nbits = 16;
base = 1000;
set = Range[base / 10, base - 1] * (10 / base);
set = Union[set, set / base];
set = Union[set, 1 / set];
While[Length[set] < 2^(nbits-2), set = Union[set, set / 10, set * 10]];
Off[General::infy]
m = ⌈Length[set] / 2⌉;
set = Union[{0, 1 / 0}, Take[set, {m - 2^(nbits-3) + 1, m + 2^(nbits-3) - 1}]];
set = Union[set, -set];
Length[set]
32 768
```

This is the *Mathematica* code for generating the number system.

Notice: no “gradual underflow” issues to deal with. No subnormal numbers.

IEEE Intervals vs. SORNs

- Interval arithmetic with IEEE 16-bit floats takes 32 bits
  - Only 9 orders of magnitude dynamic range
  - NaN exceptions, no way to express empty set
  - Uncertainty grows *exponentially* in general

- SORNs with connected sets takes 32 bits
  - 10 orders of magnitude dynamic range
  - No indeterminate forms; closed under + – × ÷
  - Automatic control of information loss
  - Uncertainty grows *linearly* in general

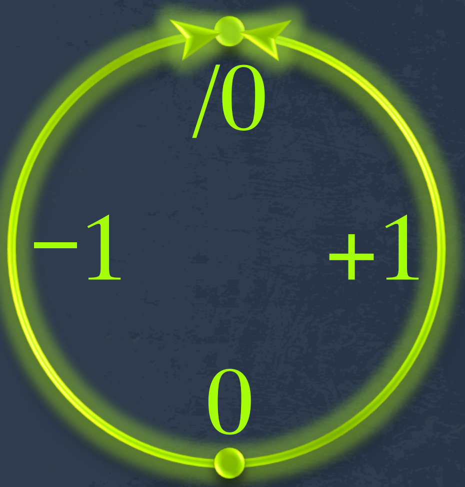
Future Directions

- Create 32-bit and 64-bit unums with new approach; table look-up still practical?
- Compare with IEEE single and double
- General SORNs need run-length encoding.
- Build C, D, Julia, Python versions of the arithmetic
- Test on various workloads, like
    - *n*-body
    - ray tracing
    - FFTs
    - linear algebra done right (complete answer, not sample answer)
    - other large dynamics problems

Summary

A complete break from IEEE floats may be worth the disruption.

- Makes every bit count, saving storage/bandwidth, energy/power
- Mathematically superior in every way, as good as integers
- Rigor without the overly pessimistic bounds of integer arithmetic



This is a shortcut to exascale.