Boundary Number Systems -- Conway

William Bricken

January 2001

Conway Numbers (Surreal Numbers)

John Conway (and later Don Knuth) constructed all known types of numbers from the simplest possible beginning, *making a distinction in the void*. The generative definition is

> A ***number*** is a partitioned set of prior numbers, `{L|G}`,
> such that no member of `L` is greater than or equal to any member of `G`.
>
> The initial number is when `L` and `G` are both void: `{ | }`

The set `L` contains *lesser* numbers, while the set `G` contains *greater* numbers. Both `L` and `G` can be *void*, that is, they can be collections without any members.

Let `xL` be an arbitrary member of `L`, and `xG` be an arbitrary member of `G`.

```
x = {xL|xG}        such that no xL >= any xG

                        i.e. every xL < every xG
```

By definition, `no xL >= any xG` is true whenever `L` is empty, even if `G` is empty. When there are no members of `L`, every member is less than any in `G`.

Partitioning Nothing

"Before we have any numbers, we have a certain set of numbers, namely *the empty set*, `{}`."
-- John H. Conway

Base: `{ | }` empty partitions of the empty set

Generator: every partition of the set of prior numbers

The empty set is not the base of the system, rather the *act of partitioning* is the base. Partitioning creates the first distinction, which serves as sufficient structure to build all numerical forms and operations.

The conventional names of numbers can be assigned to Conway numbers. For example:

```
{ | } = 0
```

We can test if this first partition is a number:

Is `{ | }` a number?

`every xL < every xG?` yes since there are no `xL`

Ordering and Equality

We next define the ordering of numbers. A Conway number is defined by comparing the members of each partition. Ordering and equality are defined by comparing all the members in a partition of one number to the *value* of another number (not to its partitions).

Two Conway numbers are ***ordered***

```
x >= y        when        no xG <= y and no yL >= x

                          i.e. every xG > y and every yL < x
```

Example: let `x = {0,1|2,3}` and `y = {-1|1}`.

To determine the ordering, we will need to know the value of each of these numbers. To be shown later, `x = 1 1/2` and `y = 0`.

```
Is x >= y?

      every xG > y            xG = {2,3}, y = 0        true
      every yL < x            yL = -1, x = 0           true

      therefore {0.1|2,3} >= {-1|1}
```

Two Conway numbers are ***strictly ordered***

```
x > y         when        x >= y and not y >= x

                    i.e.  all xG > y, all yL < x, some xL < y, some yG > x
```

Two Conway numbers are ***equal***

```
x = y         when        x >= y and y >= x

                    i.e.  all xG > y, all xL < y, all yL < x, all yG > x
```

Example:

Is `{ | } >= { | }` is `0 >= 0?` `x=0` `y=0`

`every xG > 0 and every yL < 0?` yes since there are no `xG` or `yL`

`By symmetry y >= x, thus 0 = 0`

Now we will determine how to find the conventional value of a Conway number, and how to identify the canonical form of a Conway number.

Building from Zero

`0` is a Conway number, making the set of numbers currently known = `{0}`. This generates three new number partitions:

```
{0| }        { |0}        {0|0}
```

`{0|0}` is not a number, since there is an `xL >= xG`, namely `xL=0, xG=0`

`{0| }` is a number, call it `1`

`{ |0}` is a number, call it `-1`

What is the ordering of these new numbers? For illustration, we'll test `0` against `-1`:

Ordered:

Is `{ | } >= { |0}`? i.e. is `0 >= -1`? `x=0, y=-1`

`every xG > -1 and every yL < 0?` yes since there are no `xG` or `yL`

Thus `0 >= -1.`

Strictly ordered:

Is `{ | } > { |0}`? i.e. `not(-1 >= 0)`?

`x= { |0} = -1` and `y = { | } =0`

```
every xG > y?          xG = { }, y = 0         true
every yL < x?          yL = { }, x = -1        true

-1 >= 0, therefore not(-1 >= 0) is false,
{ | } > { |0} is false.
```

Thus `0 > -1`. Similarly (tests omitted) `1 > 0.`

Later, we will see that `{0|0}` is a Conway *imaginary* number.

Building from One

Now, the current set of prior numbers = `{-1,0,1}`, with a strict ordering, `1 > 0 > -1`.

Three prior numbers generate 8 (2^3, the powerset) sets to form partitions with. The definition of a number constrains the forms generated from these sets to 21 new number forms:

`{-1|0}   {-1|0,1}   {-1|1}   {0|1}   {-1,0|1}   { |R}   {L| }`

where `R` and `L` stand for any of the eight sets in the powerset of prior numbers.

Conway numbers have multiple representations, just like 3+4 is an alternative representation of 7. A closer analogy would be to have a number which is written in different languages (three, trois, drei,...). For example:

`0 = { | } = {-1| } = { |1} = {-1|1}`

In general:

the smallest `xG` defines `G`, the largest `xL` defines `L`.

This is easy to see since the tests for numbership and ordering are of the form `All xG > ?` and `All xL < ?`. If every number in a set is larger/smaller than a particular number, the smallest/largest member characterizes the set.

The new numbers are:

`{1| } = 2`   `{ |-1} = -2`   `{0|1} = 1/2`   `{-1|0} = -1/2`

This gives a hint about how to think about Conway representations: the new number is the "between" of the largest `xL` and the smallest `xG`. When one side of the partition is void, a new integer is formed.

Number Forms

How do we know what conventional number corresponds to each Conway number? In general:

If there's any number that fits, then use the simplest number that fits.

That is, given a number `{a,b,c,...|d,e,f,...}`, the interpretation of that form is the simplest conventional number which is strictly greater than `max[a,b,c,...]` and strictly less than `min[d,e,f,...]`.

A contribution of Conway numbers is that they incorporate all types of numbers in one consistent system.

Let `n` be the maximal element on the Lesser side of a number when it is on the Lesser side. Let `n` be the minimal element on the Greater side when it is on the Greater side.

`x` is an ***ordinal*** number when

`x = {L| }`

`{n| } = n+1`

`x` is a ***negative integer*** when

```
x = { |G}

{ |-n} = -(n+1)
```

`x` is a ***fraction*** when

```
{n|n+1} = n + 1/2

{0|2^-(n-1)} = 2^-n

{p/2^n | (p+1)/2^n} = (2p+1)/2^(n+1)
```

`x` is a ***real*** number when

```
x = {x - 1/n|x + 1/n}   for n > 0
```

`xL` is arbitrarily close to `x` from the bottom, and `xG` is arbitrarily close to `x` from the top.

Conway Operators

For a representation to be useful, it must be accompanied with a complete set of transformation rules. Here, the standard numerical operations are defined recursively for Conway numbers:

Addition

Base: `0 + 0 = { | }`

Generator: `x + y = {xL+y, x+yL | xG+y, x+yG}`

Example: `2 + (-1) = {1| } + { |0}`

```
xL+y = 1 + (-1)    = 0        this sum is computed recursively
x+yL = 2 + void    = void
xG+y = void + (-1) = void
x+yG = 2 + 0       = 2        this sum is computed recursively

x + y = {0|2} = 1
```

To show that `{0|2}` is a representation of `{0| } = 1`, show equality:

```
x={0|2} =?= y={0| }

every xG > y                  2>1  true
every xL < y                  0<1  true
every yL < x                  0<1  true
every yG > x                  none true
```

Negation

Base: `-0 = { | }`

Generator: `-x = {-xG|-xL}`

Changing signs reverses the location of each partition.

Multiplication

Base: `0*0 = { | }`

Generator:

```
x*y = {xL*y+x*yL-xL*yL, xG*y+x*yG-xG*yG |
       xL*y+x*yG-xL*yG, xG*y+x*yL-xG*yL}
```

Multiplication recurs on each partition of each variable.

Division

`y` is a number and `x*y = 1`

Base: `y = {0| }`

Generator:

```
y = {0, (1 + (xG-x)*yL/xG, (1 + (xL-x)*yG/xL |
        (1 + (xL-x)*yL/xL, (1 + (xG-x)*yG/xG}
```

Infinities and Infinitesimals

Conway numbers allow computation with a diversity of infinities and infinitesimals. ***Infinite*** numbers are generated when an infinity of ordinals is included in xL:

`w = {0,1,2,...| }` `w` is infinite

Unlike conventional numbers, operations on varieties of infinite numbers are defined:

```
w + 1   = {0,1,2,...,w| }

w - 1   = {0,1,2,...|w}

w/2     = {0,1,2,...|w,w-1,w-2,...}

w^(1/2) = {0,1,2,...|w,w/2,w/4,w/8,...}
```

Imaginary Star

The form {0|0} is not a number. However, it can be treated as an *imaginary* number, *, such that

```
* + * = 0                    * =/= 0
```

Star is its own additive inverse.

```
* = -*
```

Generally,

```
n + * = {n|n}  for any n

n + * = {0+*,1+*,...(n-1)+*|0+*,1+*,...(n-1)+*}
```

Consider {0|*}, which is less than or equal to {0|1},{0|1/2},{0|1/4},.... This number is infinitesimally close to 0.

> {0|*} is a positive number which is smaller than all other positive numbers, call it d+.
>
> {*|0} is a negative number which is larger than all other negative numbers, call it d-.

```
{d+|d-} = {d+|0} = {0|d-} = {0|0} = *

d+ + * = {0,*|0}

d- + * = {0|0,*}
```

Later we will show that the same mathematical concept shows up naturally in the James calculus.

Commentary

Conway is an acknowledged mathematical genius, and Conway numbers are generally thought to contain some profound insights. To date, very few people know how to find utility in this form, although it appears that *comparison* of Conway numbers (ordering, equality) alone provides a significant set of tools for analysis. The advantages in representation are paid for by having the computability of standard operations {+,-.*,/} that are more complex. However, even the relevance of standard operations is in question for Conway numbers.

The system contains no inconsistencies or singularities; in fact it removes many inconsistencies (division by zero, incrementing infinity) in conventional numbers.

References for further study:

J.H. Conway (1976) *On Numbers and Games*, Academic Press

An academic introduction to Conway numbers. Contains some of the material in the next reference, but presented more succinctly.

E.R. Berlekamp, J.H. Conway and R.K. Guy (1982) *Winning Ways* (two volumes), Academic Press

An extensive primer on Conway numbers phrased in terms of adversarial games such as tic-tac-toe and nim. Requires study.

D.E.Knuth (1974) *Surreal Numbers*, Addison-Wesley.

An introductory primer intended to be easy to understand. It did not help me much.