Boundary Number Systems -- Kauffman

William Bricken

January 2001

Kauffman Numbers

This system assigns a different interpretation to containment. Depth of containment is a type of place notation, each container multiplies its contents by two. The depth of nesting of each container indicates the power to which the base is raised.

Kauffman boundaries can have any base, the value of the base specifies how to interpret a form. We have selected the base two in order to show a similarity to binary numbers.

Containment does not have the same constraints as place notation, in that each container is independent of the rest. There is no reliance on the position of a digit in a form. Using conjunction in space to represent addition leads to a naturally parallel representation.

Each number has many representations depending upon the degree to which it has been condensed from stroke notation to container notation. For example, some of the ways we can write the number six are:

```
******     **(*)**     (*)(*)(*)     ((*))(*)     ((*)*)
```

We thus have two canonical ways to represent numbers, as only strokes (hard to work with) or as maximally contained (easy to work with). The computational effort in Kauffman numbers is in converting into maximally contained forms. Addition and multiplication are constant time operations. Thus the trade-off in this system is that operations take almost no effort, but results are either hard to read (stroke notation) or take effort to standardize into maximally nested forms.

Kauffman Integers (String Version)

```
1     *
2     **     ->   (*)
3     ***    ->   (*)*
4     ****   ->   (*)(*)   ->   ((*))
```

In reading a Kauffman number, all forms which share a space are added together. A boundary doubles the value of its contents. Thus the readings of the number six above are:

Kauffman	Decimal
******	1+1+1+1+1+1
(*)	1+1+2+1+1
(*)(*)(*)	2+2+2
((*))(*)	2(2)+2
((*)*)	2(2+1)

Each star is a unit, each container delineates an order of magnitude in the chosen base. The notation is not in a particular base, since added contents may sun to more than that base.

Canonical Transformations

The canonical form of a number is the one with the least number of stars and boundaries. These transforms convert numbers into a canonical form.

Name	*Instance*	*Algebraic*	*Interpretation*
Commutativity	`*A = A*`	`A B = B A`	`A+B = B+A`
Power	`** = (*)`	`A A = (A)`	`A+A = 2A`
Distribution	`(*)(*) = (**)`	`(A)(B) = (A B)`	`2A+2B = 2(A+B)`

Commutativity and associativity are built into the notation. However, commutativity must be an explicit rule in an implementation of the string version. The power transform converts the sum of two identical forms into one form doubled. Distribution collects doubles.

Operations

Computation via addition and multiplication is strictly algebraic. Addition is sharing a space, putting a collection of forms into a common space. Multiplication is substituting a form for each occurrence of a unit in another form. Unlike Spencer-Brown numbers, the interpretation of addition applies to all spaces, regardless of depth of nesting. This permits deeply nested representations, with each level interpreted as a multiplication by a base. Thus depth takes the place of sequence in representing numbers in a power-based notation.

```
Addition          A+B      juxtapose           A   B

Multiplication    A*B      substitute for *    A   B
                                               \__|
```

The connecting arc (__|) means

*substitute one form for every occurrence of * in the other form.*

Example `2(4+3) = 14`

Substitute `(*)` for `*` in `((*))(*)*`:

```
(*)     ((*))(*)*    =    (((*)))((*))(*)
   \______|___|_|
```

This form reduces to a canonical version:

```
(((*)))((*))(*)
(((*)    *)  *)
```

Reading the result from the innermost star:

```
(((*)*)*)          2*(2*((2*1)+1)+1) = 14
```

Here is the symmetrical case: Substitute `((*))(*)*` for `*` in `(*)`:

```
(*)    ((*))(*)*    =    ( ((*))(*)* )
 |___/
```

Reducing to canonical:

```
(((*))(*)*)
(((*)  *)*)
```

Inverse   Operations

Kauffman numbers include a standard version of the additive inverse. Notationally, Kauffman uses an overbar; here we will use the traditional minus sign.

The negation operation is achieved by multiplication by minus one.

```
-A = (-1)*A

-A =def= substitute -* for A in the form
```

We will use a period, **.**, as a notational tool to disambiguate minus signs in front of entire forms. Thus:

```
-3          -.(*)*. = (-*)-*
```

The new rules (and extensions to existing rules) to handle negative numbers are:

Name	*Rule*	*Interpretation*
definition	`-*`	`-1`
cancel	`*-* = -** = void`	`(1 + -1) = (-1 + 1) = 0`
double	`(-*) = -*-*`	`2(-1) = -1 + -1`
minus-minus	`--* = *`	`-(-1) = 1`

These two rules permit negative numbers to migrate across doubling boundaries:

compensate	`(A)* = (A *)-*`	`2A + 1 = 2(A+1) - 1`
subtract	`(A)-* = (A -*)*`	`2A - 1 = 2(A-1) + 1`

Division is handled by a new type of boundary, the inverse of the doubling boundary.

definition	`[A]`	`A/2`
divide	`([A]) = [(A)] = A`	`2(A/2) = (2A)/2 = A`

We will not cover division here. An example of subtraction follows:

```
6 - 2

((*)*)-(*)          transcribe
((*)*)(-*)          negation
((*)*  -*)          distribution
((*)     )          cancel
   4                interpret
```

Kauffman Arithmetic (Molecular Version)

Addition is *physical mixing*. 4+3

```
  *    *              (*)       (**)            ((*))
*   *       -->   (*)   *    -->       *   -->           *
  *   *   *         (*)          (*)               (*)
```

Multiplication is *chemical mixing*. 2*7

```
     ((*))            (((*)))
   ____|                                (((*)))
  /         (*)  -->         ((*))    -->              -->  (((*)*)*)
 (*)    *    |          (*)                  ((*)*)
  \_____|____|
```

Commentary

The single container system is limited to integers.