Boundary Number Systems -- Bricken

William Bricken
January 2001

Bricken Graph-numbers

These numbers are a parallel implementation of Kauffman numbers, with some extensions.

The following presentation is in a different style than those in other sections of this document.

BOUNDARY NUMBERS specify a formal redefinition of the concept of number. Rather than being inert objects that are operated *upon*, boundary numbers are active objects that *compute themselves*. This is a fundamental refocusing of the concepts of object and operator. Rather than having easily stated and relatively useless numerical objects coupled with computation intensive operators, the boundary model is:

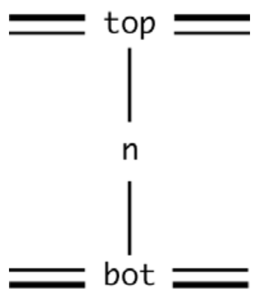
ACTIVE OBJECTS that dynamically compute their value
coupled with easily stated and relatively inert OPERATORS.

Thus, the computational effort is in finding out the value of a boundary representation of a number. Operating on boundary numbers is a trivial process; *operations are independent of the magnitude of the numbers being manipulated*.

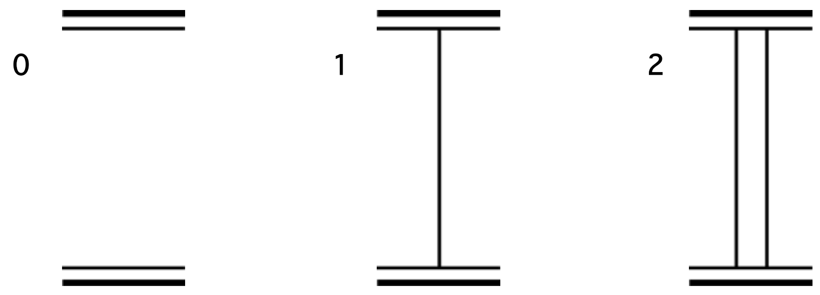
The computational trade-off, then, is in determining the value of a result. The READING process is strongly parallel and *more efficient* than traditional computation. (Reading a value is log(n), where n is the number of bits in the binary representation of the result.) And reading is required only once, when the result of any combination of operations is desired.

DEFINITIONS

A boundary number has a top and a bottom. The magnitude of the number is expressed by the connectivity between the top and the bottom.



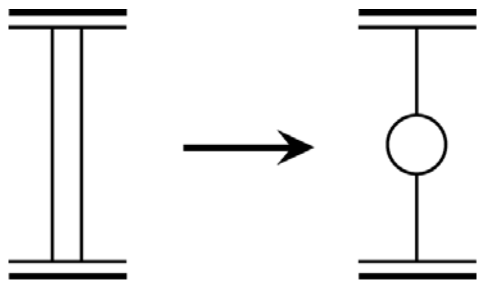
ZERO has no connectivity, ONE has simple connectivity.



TOP and BOT bound a parallel computational space. The connectivity network that represents the number settles into a standard form which is easily recognized as a linear number by TOP. The parallel standardization process is needed only at i/o time and is independent of computation across numbers. Alternatively, standardization can be replaced by a reader which sweeps the connectivity network to return the linear form of the number.

PARALLEL STANDARDIZATION RULES

GROUP



The GROUPing operation transforms simple connections into binary multipliers, which serve the same function as place holders in linear notation. Using an asterisk for the unit, and a parens container for the group, this rule can be written:

```
** ==> (*)
```

Grouping can be generalized to any base; the above rule is base 2. Grouping essentially converts a base-1 unit notation into a base-2 notation. The grouping operation applies to any level in the boundary number, not just between TOP and BOT.

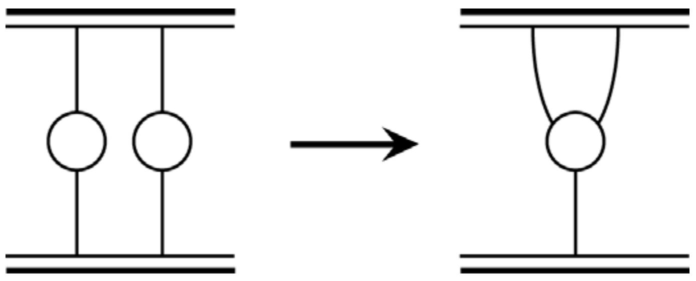
Algebraically, Grouping is a variant of the distributive law. In the simplest form:

$$1 + 1 = 2$$

And more generally:

$$n + n = n(1 + 1) = 2n$$

COALESCE



COALESCE reduces redundancy. Using asterisks and parens, this rule can be written:

```
(*)(*)  ==>  (*  *)
```

In a shorter, void-based notation, this is:

```
 *)(*   ==>   *  *
```

Or more simply:

```
  )(    ==>  <void>
```

Coalesce generalizes to any number of nodes, it is independent of a base. The coalesce operation applies to any level in the boundary number, not just between TOP and BOT.

Algebraically, Grouping is also a variant of the distributive law. In the simplest form this is:

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

And in the general form, it is:

```
 2n + 2n   =  2(n + n)
```

Note that, from the perspective of algebraic rules, grouping and Coalesce are different forms of the same distributive rule.

MULTIPLE REPRESENTATIONS

The same boundary number has many different representations (networks of connectivities, see example below). A boundary number reader would return the same conventional number for each representation. The application of Grouping and Coalesce is a standardization method. *The standardization process minimizes the effort of reading*.

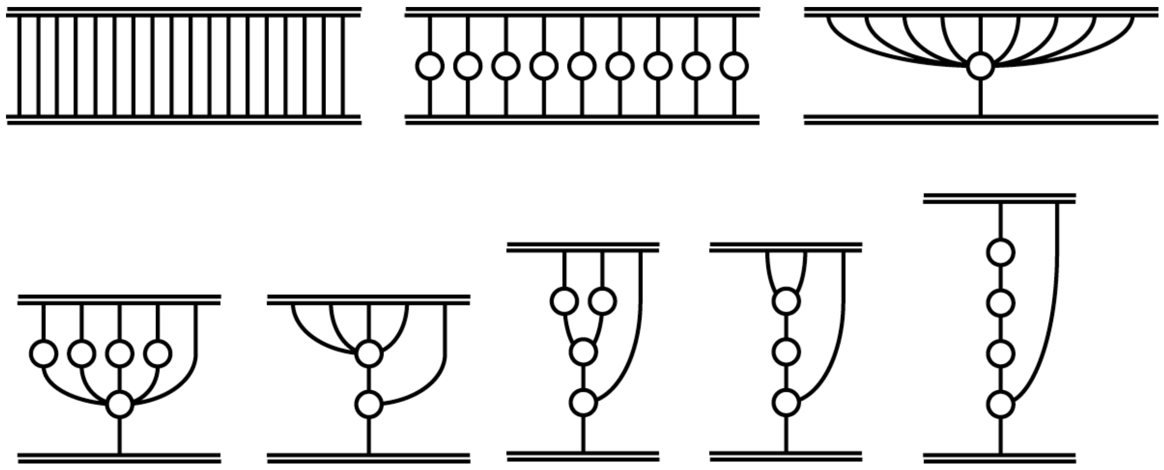
The standardization rules can be used without TOP or BOT, because

every node is top to its lower neighbors
and
every node is bot to its upper neighbors.

Therefore the standardization process can take place IN PARALLEL between any two nodes.

An example of multiple representations and the sequence of boundary number standardization steps is the number 18:

18



These forms correspond to binary partitions of conventional numbers. For the above, the conventional notation would be:

```
1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1    2+2+2+2+2+2+2+2+2    2(1+1+1+1+1+1+1+1+1)

2(2+2+2+2+1)   2(2(1+1+1+1)+1)   2(2(2+2))+1)   2(2(2(1+1))+1)   2(2(2(2*1))+1)
```

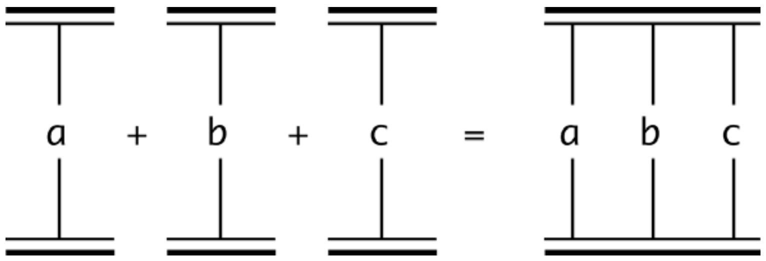
The final standardized form can be read as a binary number, each level contributes one place in the place notation system.

NUMERICAL OPERATORS

Operators in boundary arithmetic are cut and paste. Changing a pointer is sufficient to perform any elementary computation.

ADDITION

Addition is defined as merging tops with tops and bots with bots. With boundary numbers, this is simply joining tops with other tops, and bots with other bots.

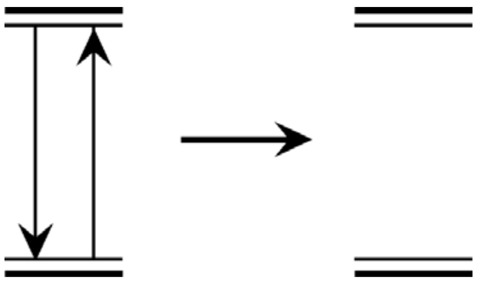


SUBTRACTION

NEGATIVE numbers are defined by the bot to top *gradient*. Numbers pointing up (bot to top) are positive. Numbers pointing down (top to bot) are negative.

Subtraction is addition of numbers that have gradients. The standardization rule that achieves subtraction is:

PLUS-CANCEL

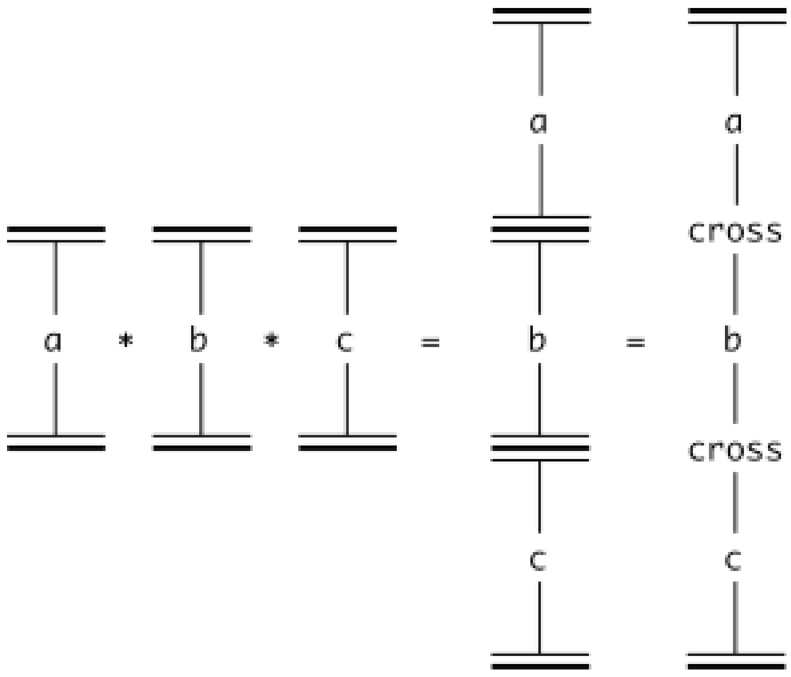


Characteristic of void based transformations, this rule permits structure to disappear.

MULTIPLICATION

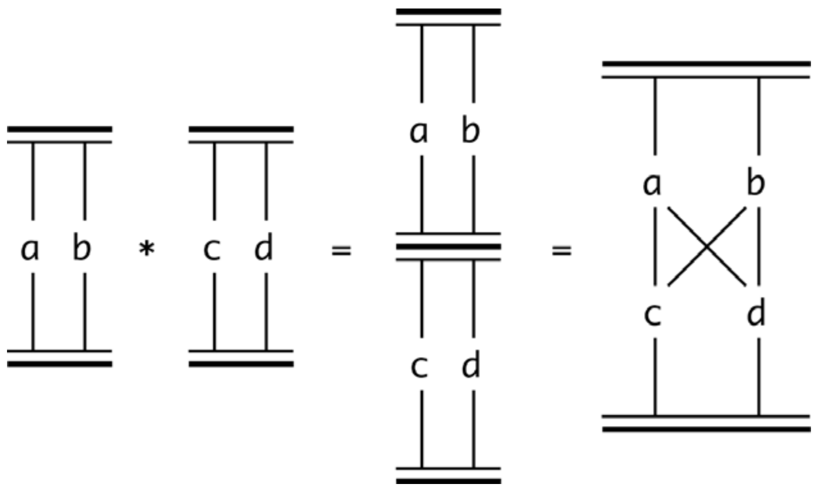
Multiplication is defined as merging tops with bots. With boundary numbers, this is a simple stacking operation.

To remove the top/bot bar which achieves multiplication, CROSS-CONNECT the bot connections with the top connections. Performing cross-connection in reverse defines FACTORING.



CROSS-CONNECT

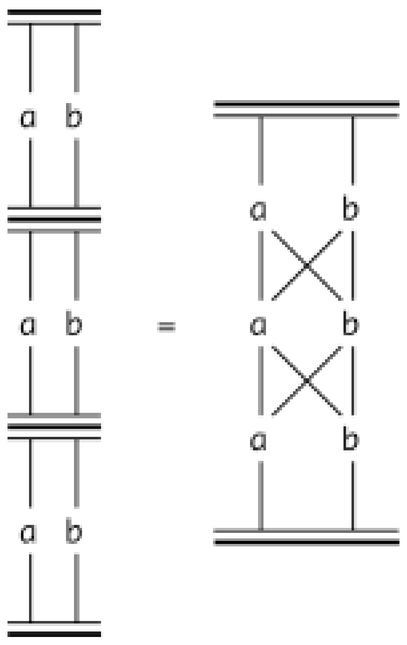
(a+b)(c+d) = ca + cb + da + db



The "X" in the last representation is a cross-connection. Reading a boundary number involves traversing all available paths; above there are four. Cross-connect is yet another version of the distributive law. The expansion from factored to polynomial form can be traced by the following boundary forms:

Consider the (beautiful) representation of the binomial theorem in boundary notation:

`(a + b)^3 =  a^3 + 3a^2b + 3ab^2 + b^3`



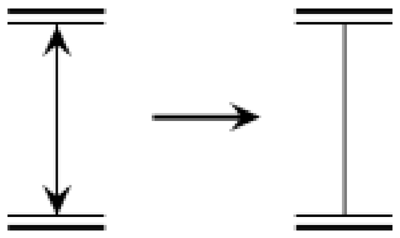
The right-hand-side represents eight paths from bot to top. One path passes through three *a* forms; three paths pass through two *a* forms and one *b* form. Similarly, three paths pas through two *b* forms and one *a* form, and one path threads through all three *b* forms.

DIVISION

RECIPROCAL numbers are defined by a gradient. The principle is the same as subtraction, but the multiply/divide gradient is recorded as a different, separate gradient than the add/subtract gradient.

The standardization rule for division is:

MULTIPLY-CANCEL

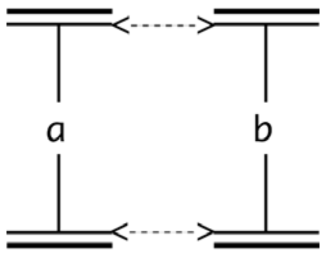


Opposite division gradients reduce to simple connectivity.

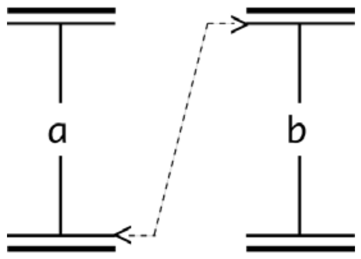
STACKING

Parallel standardization takes care of the evaluation of the form of the final value of a computation. The computation itself is achieved merely by switching pointers to either top or bot. Abstractly:

(a + b)



(a * b)



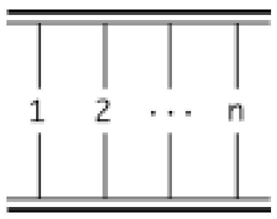
Stated simply:

To ADD: stack boundary numbers horizontally.
To MULTIPLY: stack boundary numbers vertically.

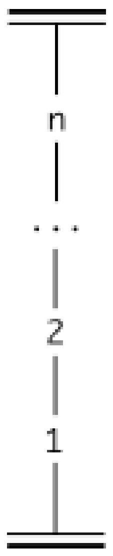
Boundary numbers require almost no effort to apply operators (addition and multiplication), almost all computational effort is in reading the numbers.

Stacking leads to a parallel redefinition of conventional SUM and PRODUCT:

SUM[1..n]



PRODUCT[1..n]



PEANO AXIOMS FOR ARITHMETIC

The four Peano axioms for the construction of arithmetic are shown in boundary notation at the end of this paper. Here are some observations about their spatial form:

1. Induction is not needed as a reasoning axiom. Instead it is subsumed by the parallel process of standardization. For boundary variables to standardize, connectivities that represent magnitude must be decomposed pictorially (by running the standardization rules backwards). The decomposition steps achieve induction, but are more efficient.

2. The concept of a successor function is not really needed either. The cut and paste definitions of boundary + and * are a sufficient axiomatization of the operators. For the addition operation, the successor function is confounded with parallel standardization of representations in the same space. In multiplication, the successor is confounded with cross-connection of stacked representations.

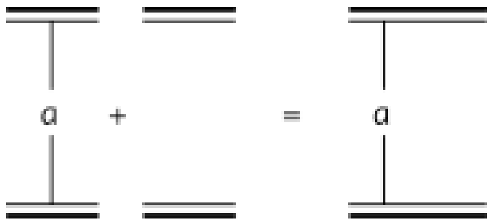
3. The zero axioms are quite unnecessary. The marvelous characteristic of void based representation is that the SYMBOL OF NOTHING is replaced by a LITERAL NOTHING. During computation, the halting condition is *nonexistence* of connectivity rather than the identification of a special token for bottom (i.e. "0").

4. The final rewrite in Axioms III and IV illustrates the similarity of boundary notation to the linear notation for Peano's definitions. They are not necessary within the network connectivity formalism.

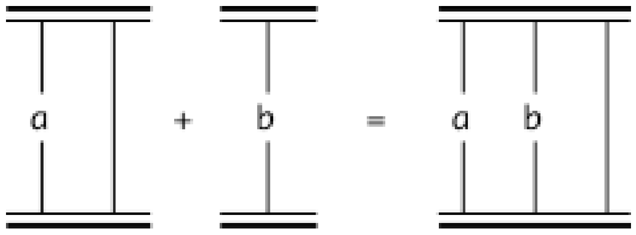
5. The general rule of parallel boundary operations is: *recording the problem is sufficient to generate the answer.* All effort is in reading the result, and this need be done once at the exit to computation.

PEANO AXIOMS IN BOUNDARY FORM

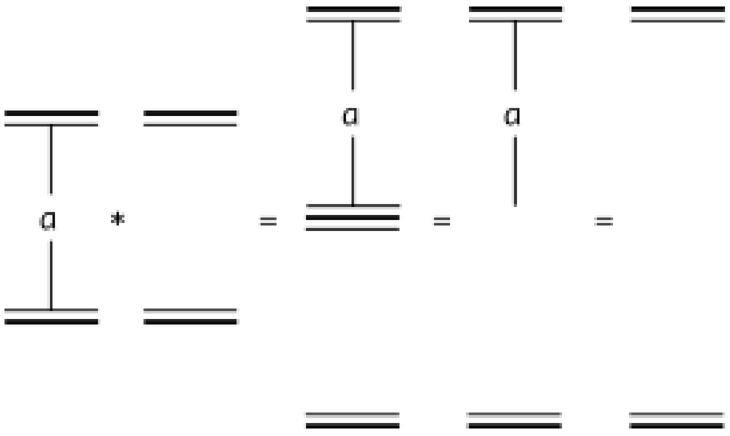
I. **a + 0 = a**



II. **a' + b = (a + b)'**



III. **a * 0 = 0**



IV. **a'*b = (a * b) + b**