BOUNDARY ALGEBRA

Jeffrey M. James with William Bricken
May 14, 1993

Abstract

Boundary algebra (BA) is a mathematical alternative to standard algebra. It represents numbers and functions using space and boundaries, which can be manipulated with three simple equivalence rules. In BA, the behavior of large constructions, such as complex numbers and transcendental functions, can be directly deduced from the basic rules of BA, suggesting that it represents a fundamental core of elementary algebra.

Introduction

How we communicate ideas constrains how we think about them. We communicate mathematical ideas with notations which bias our thoughts about mathematics and influence our beliefs about the foundations of mathematics. Although notation plays a seemingly transparent role in communicating mathematics, it tends to carry many epistemological assumptions which may be inappropriate in the long run.

In this paper, I suggest that we break some epistemological assumptions. I describe a radically different way of communicating some seemingly well-understood mathematical concepts, revealing a greater simplicity to them than is commonly known. I present boundary algebra (BA) as an alternative representation of numerical algebras. BA is a radical approach to representation because it uses spatial constructs rather than linear ones. And BA clarifies numerical concepts because it is simple in design yet extensive in scope.

I first introduce the notion of spatial representation. Then I formally define boundary algebra and map it to traditional numerical algebras, drawing important distinctions between how the two represent mathematics. From this comparison I conclude that boundary algebra conveys more insightful, more fundamental aspects of numerical algebra than does the traditional notation and that it may be time to break those epistemological assumptions.

Conventions

I must note here that the form of description you are reading is principally linear and that in many cases I cannot avoid the linear constraints of this medium when describing spatial forms. Artifacts of the description should be recognized as independent of the forms themselves. In the cases where this influence is particularly strong but unavoidable, I relate the source of the

problem and suggest other media which may more accurately portray the forms.

One such unavoidable problem deals with equivalence. An equivalence relation states that certain items lack any meaningful differentiation in the system of discussion. Any item can substitute for any other in the equivalence class and be suitable in all cases. A linearity problem arises when describing an equivalence because forms must be identified but neither should be given any strict preference. A seemingly harmless "X=Y" carries an expectation that the Y is the simpler element, an unruly rhetorical corruption of the description. Nevertheless, the equals sign will be used to denote equivalence.

Independent of symmetry criteria, the equals sign does not clearly delimit the boundaries of the two spatial forms being equated. The spatial extent of each form should be recognized as perceptually bound by the typographical context including the equals sign itself, as if it were a dividing line separating two forms. The more suitable form of "X|Y" would be adapted were it not for the exceedingly dominant expectation for "X=Y".

I am also forced to adapt a standard textual description for the spatial forms. When items in space are listed, as "a b", they should be imagined as free in space without linear constraint. When boundaries are denoted textually, as paired delimiters such as "( )", they should be imagined as closed boundaries which connect beyond the linear typography. I have introduced a single exception to the paired delimiters notation in which an empty parentheses boundary, appearing round, is written in shorthand as a small circle "o".

DEFINITION

Space

Boundary algebra differs significantly from traditional numerical algebras because it is uses space and topological enclosure as its principle constructs rather than linearity and adjacency.

Space is more compatible with mathematical semantics than linearity because linearity imposes ordering constraints on a representation even when the mathematical relationship lacks order. Compare the linear form of addition, "a+b+c", with a spatial representation of the same, shown in Figure 1. The spatial form leaves the items unordered--a faithful interpretation of the combination. The linear form must overcome its falsely imposed constraint by introducing commutativity rules,

$$a+b=b+a$$

and associativity rules,

$$a+(b+c)=(a+b)+c$$

to counteract linear constraints. In space, the items are inherently unordered because ordering has not been imposed upon them.

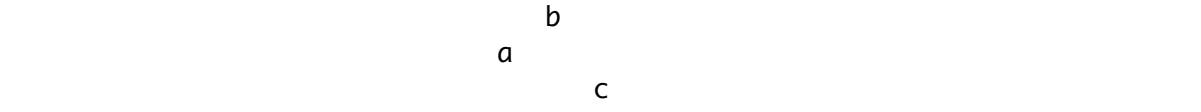

Figure 1.

Space further adheres to mathematical semantics because it comes with an identity element built-in. All mathematical functions characteristically have an identity element which leaves a result unchanged when a function is applied with it. For instance, the identity element for addition is zero. The relation describing the additive identity is written linearly as

$$X + 0 = X$$

but written spatially as

$$X = X$$

The spatial representation trivializes the identity because the identity element for spatial addition is empty space; adding nothing does not change a spatial collection.

Making use of empty space in this manner leads to a more complicated dependency on *void substitution*. Empty space is called *the void* and when something equivalent to the void is reduced to it or introduced from it, this substitution is called void substitution.

A principle case of void substitution occurs when two elements in space cancel each other out. For example, if "a b" equals the void then the expression shown in Figure 1 can be reduced to just "c" by substituting void for "a b". Likewise, the equivalence goes the other way so that an additional "a b" could be introduced to the expression by substituting for the void somewhere in the open space.

The one constraint on void substitution is that what is taken or introduced must conform to spatial *boundaries*. Boundaries delimit space. A void substitution must involve a clearly delimited space: when introducing "a b" from the void, both elements must appear on the same side of every boundary.

They can be introduced together on one side or the other but not across the boundary. Some legal and illegal examples of void substitution are shown in Figure 2.

Legal	*Illegal*
( ) = a b ( )	( ) != a (b)
( ) = (a b)	( ) != (a) b
(()) = (a b())	(()) != a(b())
(()) = ((a b))	(()) != ((a)b)

Figure 2. Void substitution with "a b = ".

Boundaries provide control over space. They create perspective by framing a space for observation, by forming the edges of objects, and by distinguishing objects out of an environment. Each of these uses of boundaries arises in Figure 2. The space of the figure itself provides a perspective, framed by the surrounding white space. The space of each expression is confined by line spacing and separation between the two columns. The space of symbolic objects, such as "a" or "b", implicitly ends at the edge of the symbol. The only explicit boundaries in the entire figure are the parenthetical boundaries, which comprise the formal distinctions of boundary algebra.

Boundaries are fundamental to representation. They are used extensively to parse spatial as well as linear configurations. By utilizing them explicitly, boundary algebra seeks to control this basic representational mechanism. Boundaries are identified so that this distinction carries semantic value in addition to syntactic structure.

I have just introduced space using undefined elements "a", "b", and "c" and an undefined boundary. I will now define the elements which comprise boundary arithmetic and introduce variables to make an algebra.

Elements

The boundary elements are composed entirely of the void and of three types of boundaries.

Definition. Let P denote the collection of all well-formed boundary expressions using three pairs of bracket characters denoting left and right edges of their respective boundary types: <>, (), []. P is defined recursively by two rules:

1. If a1, a2, ... an (n>=0) belong to P,
   then the unordered collection a1 a2 ... an belongs to P.

2. If b belongs to P, then <b>, (b), and [b] belong to P.

The first rule defines the principle of *spatial collection*: Collecting elements of P together forms an element of P. Spatial collections are unordered and flat so that when two or more collections are brought together, the resulting collection looses the individual containments of the separate collections.

The collection principle also defines the initial element of P: since there are no initial elements of P, the first rule can only define the empty collection known as the void. The void is present in any and all spatial collections an infinite number of times (there is empty space everywhere). Spatial collection by itself can define only the void because it cannot construct any other element; collections of the void are still void.

The second rule defines the principle of *spatial distinction*: Bounding an element of P forms an element of P. Spatial distinction initially creates three elements from the void using each of the three boundaries: <>, (), and []. Although space is unordered, boundaries add a constraint that prevents rearrangement across the boundaries. Because boundaries can surround other boundaries, they create a hierarchy of nested space.

From the void, spatial distinction creates the following elements:

<>, (), [], <<>>, <()>, <[]>, (<>), (()), ([]), [<>], [()], [[]], ...

These elements can be spatially combined to create more elements:

<><>, <>(), <>[], ()<>, ()(), ()[], []<>, []() , []<>, ...

The results of these combinations can be further distinguished and combined (recall that "o"="()"):

<oo>, ([oo][ooo]), [oo], (([[<o>]] <[oo]>)), ...

And so on, and so on...

Equivalences

These forms would not be significant were they all equally unique. The meaning derives from equivalences of the various forms, defined as follows.

Definition. Let R denote the set of equivalence classes of elements of P under the equivalence relation generated by the following equivalences, valid for all A, B, C in P:

1. A <A> =	**Inverse Cancellation**
2. ([A]) = A = [(A)]	**Involution**
3. (A [B]) (A [C]) = (A [B C])	**Distribution**

Inverse Cancellation. The first of these rules defines the *inverse boundary*, denoted around element A as <A>. Element A of P combined with <A>, the identical element distinguished by the inverse boundary, equals the void. Put simply, a combination of something and its inverse cancels out. Every element has an inverse under space combination because it can be created merely by enclosing the element within the inverse boundary.

Involution. The second equivalence rule defines the symmetry between the *instance boundary* and the *abstract boundary*, denoted around element A as (A) and [A], respectively. Involution maintains that the distinctions made by these boundaries cancel out: instance around abstract around element A equals A itself and abstract around instance around element A also equals A itself.

Distribution. The third equivalence rule defines the relationship between instance and abstract. In the distribution form, these two boundaries allow A to modify B and C. Distribution states that this modification is equivalent whether it modifies them together or separately. Distribution can be visualized as a cell division or cell union where the A is part of the cell wall while B and C are inside of the wall.

Variables

The equivalence rules define how the elements of boundary arithmetic can be transformed. A full fledged algebra further requires *variables* which stand for elements of this arithmetic. These variables will be denoted using alphabetic characters.

These variables should not be confused with the template variables used in the equivalence rules, though the two appear similar. The template variables are capitalized to show that they represent any and all elements in the boundary arithmetic; each equivalence holds for all possible replacements of these variables. Algebraic variables differ in that they represent unknowns which may or may not exist, rather than representing everything possible.

Without algebraic variables, boundary constructions are just elements of the

arithmetic, as defined above. With algebraic variables, boundary constructions cover algebraic expressions in addition to the arithmetic elements. The term "boundary expression" refers to a configuration of boundaries that possibly includes algebraic variables.

To use an equivalence rule, first replace each template variable with some boundary expression. Replace a template variable identically throughout the rule, though different variables can be given different values. With all template variables replaced, the rule specifies an direct equivalence between boundary expressions. Then match either side of this equivalence to part of a larger expression and substitute the other side of the equivalence rule for matched expression. Replace template variables so that the resulting rule matches part of the working expression and causes a useful substitution into a preferred form.

==========

Inverse: A <A> =

```
oooo<oo>
oo                                  A = oo

([ooo]          )
([ooo][oo]<[oo]>)                   A = [oo]

<<(x y)>>
<<(x y)>> <(x y)> (x y)             A = (x y)
                  (x y)             A = <(x y)>
```

Involution: ([A]) =

```
([oo])
  oo                                A = oo

(  <([ooo]<[oo]>)>  )
(([<([ooo]<[oo]>)>]))               A = <([ooo]<[oo]>)>

[  (x)([(y)  (z)])]
[  (x)  (y)  (z)  ]                 A = (y)(z)
[([(x)  (y)])(z)  ]                 A = (x)(y)
```

==========

Figure 3a. Examples of applying the boundary rules.

Match and substitute is a basic transform mechanism which is easily mastered. Some examples of match and substitute with the boundary rules are given below

in Figures 3a and 3b with their corresponding variable replacements. Equivalent expressions have been stacked on each other and laid out so that the application of the rule is visually apparent. Because the rules apply equivalently in either direction, each expression in a group is equivalent to all others so that transformation need not progress from the top-down.

==========

Involution: [(A)] =

```
([ooo]    )
([ooo][()])                    A =

(([[(())]][ooo]))
(([  ()  ][ooo]))              A = ()
((       [ooo]))               A =

(  [x][([y]  [z])])
(  [x]  [y]  [z]  )            A = [y][z]
([([x]  [y])][z]  )            A = [x][y]
```

Distribution: (A [B]) (A [C]) = (A [B C])

```
([ooo][o])([ooo][<o>])
([ooo][o         <o>])         A = [ooo], B = o, C = <o>


(([[oooo]][]))
(([[oooo]][])([[oooo]][]))     A = [[oooo]], B = , C =

([x][y       y])
([x][y])([x][y])               A = [x], B = y, C = y
([x       x][y])               A = [y], B = x, C = x
```

==========

Figure 3b. More examples of applying the boundary rules.

Counting

I will now build some useful mathematical constructions from the rules of boundary algebra. Later, when the algebra has been translated to standard algebra, these constructions will be recognized as prevalent throughout algebra.

To begin with, items can be counted. Everything has a count of one:

Cardinality of One

```
  A                          Given
([A]    )                    Involution
([A][()])                    Involution
```

The empty instance boundary, (), serves as the abstract unit for counting elements in a boundary expression. In this configuration, the lone () denotes the singular cardinality of the A. Greater cardinalities can be formed using distribution by collapsing many singular cardinalities into one modification of many such units. For example, the cardinality of two, from which other cardinalities can be generalized:

Cardinality of Two

```
  A        A                 Given
([A][o]) ([A][o])            Cardinality of One (twice)
([A][oo])                    Distribution
```

These cardinality theorems apply to all instances of duplicate elements, regardless of their form or depth of nesting. Any spatial combination of two identical elements can be counted this way. For instance, the following case counts [X] within the context of the surrounding instance boundary:

```
([x][x]) = (([[x]][oo]))
```

A zero count, written as ([A][ ]) reduces to the void. One way to prove this is to introduce a non-zero count of the A and absorb the zero count into it:

Zero Cardinality

```
(A [])                       Given
(A []) (A [o]) <(A [o])>     Inverse
(A [       o]) <(A [o])>     Distribution
                             Inverse
```

In zero cardinality, the [] dominates the A and forces it into nothingness. This obliteration effect can be generalized using involution:

Dominion

```
  A []                          Given
[(A [])]                        Involution
[     ]                         Zero Cardinality
```

The Dominion effect forces a qualification on boundary algebra. Because the A is completely lost, a combination with [] cannot be reversed. In other words, [] has no inverse. Therefore, the preceding definitions must be qualified to exclude the <[]> element from the boundary arithmetic.

Phase

Before translating boundary algebra to standard algebra, I now prove some characteristics of the inverse boundary which will prevail throughout standard algebra. These characteristics give boundary algebra properties of phase space and extend its scope into complex and transcendental functions.

The following boundary construction possesses a curious property of independence from its contents. The combination of instance, inverse, and abstract boundaries, [<(A)>], can "move around" by pulling expressions from its context to the inside or by pushing its contents back to the outside. The following theorem derives this fluidity property.

Fluidity

```
[<(A B)>]                          Given
[<(A B)> (  [             ])]      Involution
[<(A B)> (A [             ])]      Dominion
[<(A B)> (A [(B)      <(B)>])]     Inverse
[<(A B)> (A [(B)])(A [<(B)>])]     Distribution
[<(A B)> (A   B  )(A [<(B)>])]     Involution
[                (A [<(B)>])]      Inverse
                  A [<(B)>]        Involution
```

Finishing off the cardinality proofs from the previous section, fluidity provides a simple proof of negative cardinality. Here, the inverse boundary is promoted to an inverted unit in the cardinality combination. In the proof, fluidity moves the [A] to the outside of the [<()>].

Negative Cardinality

<A>	Given
<([A])>	Involution
([<([A])>])	Involution
([A][<()>])	Fluidity

A similar theorem allows the inverse to move inside and change a positive cardinality to a negative one. The theorem uses A instead of [A] so that it applies more generally.

Inverse Promotion

<(A [B])>	Given
([<(A [B])>])	Involution
(A [<([B])>])	Fluidity
(A [< B >])	Involution

While fluidity is a more flexible theorem in boundary algebra, negative cardinality and inverse promotion are more easily interpreted in standard algebra and will provide a better basis for interpretation.

The fluidity construct without no contents represents a phase construct in boundary algebra. The arithmetic element [<()>] provides a form of the inverse susceptible to cardinality, which therefore can be built into various degrees of phase. This phase element shall be denoted by J.

Two Js in the same space cancel to the void.

J Cancellation

[<()>] [<()>]	Given
[<([<()>])>]	Fluidity
[< <()> >]	Involution
[()]	Inverse Cancellation
	Involution

Further properties of phase space and J will be discussed with the translation to standard algebra. It has been introduced here to show that the construction is independent of its interpretations in standard algebra.

INTERPRETATION

Now I will map boundary algebra onto standard algebra to show that it represents all elementary functions and complex numbers. Because BA is so simply defined, I conclude that the mechanisms of standard algebra are unnecessarily complex.

The map from standard algebra to boundary algebra has two independent degrees of freedom: the function of space and the base of the instance boundary.

A combination in space can be interpreted as an addition or as a multiplication. The void and the inverse boundary both derive meaning from this interpretation. Both interpretations will be used in this discussion to show how boundary algebra is independent of either of these operations.

When space is addition, the void is the additive identity and the inverse boundary performs the additive inverse. Similarly, when space is multiplication the void is the multiplicative identity and the inverse boundary performs the multiplicative inverse. These mappings are summarized in Table 1. Other interpretations work similarly but are functionally more complicated and lack direct interpretation.

	Boundary Form	*Standard Form for --Interpretation of Space--*	
		Addition	*Multiplication*
Combination	A B	A+B	A*B
Void		0	1
Inverse	<A>	-A	1/A

Table 1. Basic boundary forms mapped to interpretations of space.

The instance boundary translates to an exponential operation whose base is left open. As its functional inverse, the abstract boundary translates to a logarithmic operation to the same base. Though it is not mandatory to specify this base, some choices are nevertheless convenient. Using base two or base ten directly provides logarithms and exponents to that magnitude. However, the *natural* choice of base e provides direct translation from standard transcendental forms and so will be adapted here.

Table 2 interprets the initial arithmetic elements of boundary algebra in base e for both interpretations of space.

Boundary Form	*Standard Form for ---Interpretation of Space--- Addition*	*Multiplication*
<>	-0=0	1/1=1
()	e^0=1	e^1=e
[]	ln(0)=-inf	ln(1)=0
<<>>	0	0
<()>	-1	1/e
<[]>	-ln(0)=inf	1/0
(<>)	1	e
(())	e	e^e
([])	0	1
[<>]	ln(0)	0
[()]	ln(1)=0	ln(e)=1
[[]]	ln(ln(0))	ln(0)
[<()>]	ln(-1)	ln(1/e)=-1

Table 2. Boundary elements interpreted to base e.

In fact, most algebraic forms do not depend upon the base of these boundaries. The basic forms shown in Table 3 are all independent of the base interpretation.

Table 3 illustrates the relationship between the two space interpretations. To convert an expression from addition space to multiplication space, wrap the entire expression with the abstract boundary and wrap all variables with the instance boundary.

Number Forms

Various types of numbers can be represented in boundary algebra because it supports the functions that construct them. Number representations are actually constructions that use a variety of mathematical operations to build a form with the desired properties. These forms use functions such as addition, multiplication, division, and exponentiation. While the traditional number forms conceal these operations in their notation, boundary algebra includes no such shorthand--these constructions must be done explicitly using boundary operations.

Standard Form	*Boundary Form for ---Interpretation of Space--- Addition*	*Multiplication*
0		[]
1	o	
2	oo	[oo]
-1	<o>	[<o>]
-2	<oo>	[<o>]
1/2	(<[oo]>)	<[oo]>
3/2	([ooo]<[oo]>)	[ooo]<[oo]>
A	A	A
-A	<A>	[<(A)>] = A [<o>]
1/A	(<[A]>)	<A>
A+B	A B	[(A)(B)]
A-B	A <B>	[(A)<(B)>]
A*B	([A][B])	A B
A/B	([A]<[B]>)	A <B>
A^B	(([[A]][B]))	([A] B)
logA(B)	([[A]]<[[B]]>)	[A]<[B]>

Table 3. Arithmetic and Algebraic Forms in Boundary Algebra.

Number representations use these operations to achieve various degrees of expressiveness. In doing so, each form has its own combinatorial characteristics but in all cases the boundary forms still adhere to the three boundary rules. Table 4 gives a sampling of the common numerical forms along with the boundary interpretation of that form with space as addition.

The boundary interpretations are particularly lengthy because all of the operations that are implicit in the standard representation are completely expressed in the boundary form. From the boundary interpretation, the manipulations of these forms can be directly deduced from the three boundary rules.

When numbers of a given form are combined, the original format of the number must be recovered. The strategies and techniques for maintaining a format are generally considered to be the computational rules for that form, as in adding or multiplying fractions. Because the standard forms abbreviate and conceal the operations they use, they typically do not use intermediate forms for manipulating the values back into the original format. Boundary algebra, on the other hand, separates the combination and recovery steps by supporting intermediate forms and in doing so changes the nature of the computation.

Number Type	*Standard Form*	*Boundary Form with Space as Addition*
zero	0	
natural number	3	ooo
integer	-3	<ooo>
fraction	3/4	([ooo] <[oooo]>)
mixed number	2 3/4	oo ([ooo] <[oooo]>)
prime factors	2*3^2	([oo]([[ooo]][oo]))
base (r)	127	(r[(r[o]) oo]) ooooooo
decimal	12.7	(r[o]) oo (<r>[ooooooo])
base 2	1101	([([([o][oo])o][oo])][oo])o
scientific	5.1*r^5	([ooooo (<r>[o])] ([r][ooooo]))
engineering	510.0*r^3	([(r[(r[ooooo])o])]([r][ooo]))
irrationals	2^(1/2)	(([[oo]]<[oo]>))

Table 4. Example Number Forms.

BA can approach the conciseness of standard forms by introducing macros to shorten common elements. A convenient macro substitutes for base multiplication:

{A} === ([ooooo ooooo][A])

With such a macro, based numbers look better:

127 = {{o}oo}ooooooo

1101 = {{{o}o} }o

From the basic rules of boundary algebra, rules for this particular macro can be derived.

Macros to support other forms and relationships can likewise be defined and manipulative rules can be built for them. Macros can be defined for specific functions, as the above times-10, or they can be defined for specific quantities, such as 1,2,3,4,5,6,7,8,9, i or pi. In each case, rules can be generated for manipulating the macro based on the given rules of boundary algebra.

In this manner, the entire system of numerical algebra can be rebuilt based on a consistent, fundamental core.

Algebraic Manipulation

In this section, I derive some basic algebraic formulas to demonstrate how these formulas are made explicit by the boundary representation. Each derivation is compared to the standard form, with my comments on how each form supports the deductive activity.

When it comes to algebraic manipulation, boundary algebra differs from traditional algebra in two respects: the syntax parses more explicitly and the rules apply more generally. The following distribution proof demonstrates both of these advantages.

Theorem: (a+b)^2 = a^2+2ab+b^2

Standard Algebra

0. (a+b)^2
1. (a+b)(a+b)
2. a(a+b)+b(a+b)
3. aa+ab+ba+bb
4. aa+2ab+bb
5. a^2+2ab+b^2

Boundary Algebra

0.	(([[a b]] [oo]))	Given
1.	([a b][a b])	Cardinality
2.	([a b] [a]) ([a b] [b])	Distribution
3.	([a][a]) ([b][a]) ([a][b]) ([b][b])	Distribution twice
4.	(([[a]][oo])) ([([a][b])][oo]) (([[b]][oo]))	Cardinality thrice
5.	(([[a]][oo])) ([a][b] [oo]) (([[b]][oo]))	Involution

In standard algebra, expressions must be visually parsed to discern the precedence of one operation over another. Beyond step 1, the proof relies on precedence rules rather than parentheses to denote this dominance. In contrast, the boundary form has no precedence: the ordering of operation becomes irrelevant within the boundary structure.

Rules apply more generally in boundary algebra. For instance, steps 1 and 4 of the boundary proof use the same cardinality theorem,

A A = ([A][oo])

whereas corresponding steps 1, 4 and 5 of the standard proof use two different theorems, one for addition and one for multiplication.

The last step in the boundary proof makes the expression "2ab" purely associative, after counting "ab" separately. This step results from a clear distinction between counting and subsequent reduction, a distinction the standard form does not make at all.

This next proof demonstrates another parsing problem with traditional algebra, this time with subtraction. The subtraction operator hides the addition and appears to be associated with item it is up against, seen in the following proof.

Theorem: a^2-b^2 = (a+b)(a-b)

Standard Algebra

0. a^2-b^2
1. aa-bb
2. aa+ab-ab-bb
3. aa+ab+a(-b)+(-b)b
4. a(a+b)+(-b)(a+b)
5. (a-b)(a+b)

Boundary Algebra

0.	(([[a]] [()()]))	<(([[b]] [()()]))>		Given
1.	([a][a])	<([b][b])>		Cardinality twice
2.	([a][a]) ([a][b])	<([a][b])>	<([b][b])>	Inverse
3.	([a][a]) ([a][b])	([a][<b>])	([b][<b>])	Inverse Promotion
4.	([a][a b])	([a b][<b>])		Distribution twice
5.	([a b] [a <b>])			Distribution

The subtractions in step 2 of the standard proof, "-ab" and "-bb", actually modify the entire product and not just the first item, though this distinction is not clearly made. In the boundary form, the inverse boundary clearly surrounds the entire expression. In step 3, the inverse is explicitly moved to the "b" to prepare for distribution. This restructuring is only awkwardly supported in the standard notation because the subtraction is usually distributed directly without using the intermediate form, despite the clarification it provides. These steps hide the simplicity of distribution and inversion:

2. aa+ab-ab-bb
3. a(a+b)-b(a+b)
4. (a-b)(a+b)

Through its shorthand, the standard form has built up specialized rules for operations like distributing over subtraction. Similarly, boundary algebra has macros which build into specialized rules. The difference is that BA has clearly established basic operations from which the macros acquire meaning, whereas standard algebra has no consistent basis for defining these shorthand.

Exponents

In standard algebra, exponents are notoriously complicated because they are manipulated only by formula. Here, I derive these formulas in boundary algebra to show how simple they actually are. I do not list their corresponding proofs from standard algebra because standard algebra has no constructs for proving them syntactically.

When reading these proofs, recall these conversions from standard algebra:

Operation	*Standard Form*	*Boundary Form*
Multiplication	a*b	([a][b])
Multiplicative inverse	1/a	(<[a]>)
Exponentiation	a^b	(([[a]][b]))
Natural log	ln(x)	[x]
Log to base a	loga(x)	([[x]] <[[a]]>)

Collecting exponents in BA requires just a simple application of distribution.

Theorem: a^m * a^n = a^(m+n)

```
([ (([[a]] [m])) ][ (([[a]] [n])) ])    Given
(   ([[a]] [m])      ([[a]] [n])   )    Involution
(   ([[a]] [m                 n])   )    Distribution
```

Likewise, collecting bases under a common exponent is also just a simple application of distribution.

Theorem: (a^n)*(b^n) = (a*b)^n

```
([ (([[a]] [n])) ][ (([[b]] [n])) ])    Given
(   ([[a]] [n])      ([[b]] [n])   )    Involution
(   ([[a] [b]] [n])                )    Distribution
( ([[([a] [b])]] [n]) )                 Involution
```

The form a^m^n can be parsed in two ways, as (a^m)^n or as a^(m^n). The second is the accepted parsing because the first reduces to equally convenient form using multiplication, as shown below.

Theorem: (a^m)^n = a^(m*n)

```
(([[ (([[a]] [m])) ]] [n]))          Given
((     [[a]] [m]      [n]))          Involution
((     [[a]] [([m]  [n])]))          Involution
```

Negative exponents represent the multiplicative inverse. This property falls directly out of inverse manipulation in boundary algebra.

Theorem: a^(-n) = 1/(a^n)

```
(   ([[a]] [<n>])   )                Given
(<  ([[a]] [ n ])  >)                Inverse Promotion
(<[(([[a]] [ n ]))]>)                Involution
```

These properties of exponents derive directly out of the boundary algebra rules and theorems. They fit into a coherent whole.

Logarithms

Just as the exponential formulas cannot be syntactically proved in standard algebra, the logarithmic formulas cannot be either.

Theorem: ln(x*y) = ln(x)+ln(y)

```
[([x][y])]          Given
  [x][y]            Involution
```

Theorem: ln(x/y) = ln(x)-ln(y)

```
[([x]<[y]>)]        Given
  [x]<[y]>          Involution
```

Theorem: ln(x^y) = y*ln(x)

```
[(([[x]][y]))]      Given
  ([[x]][y])        Involution
```

The above theorems on logarithm manipulation are trivial because boundary algebra is based on logarithms. The logarithmic theorems are important because they connect multiplicative operations with additive operations. These theorems can be generalized to an arbitrary base by managing an additional division.

Transcendentals

Boundary algebra also represents transcendental functions. The boundary representations of these functions are all based on a single construction, called J: [<o>].

J possesses the curious property that it is its own space inverse. In other words, a pair of Js cancels to the void:

J-Cancellation: [<o>] [<o>] = J J =

This phenomena can be understood by interpreting J in standard algebra. J equals ln(-1) when space is addition and J equals -1 when space is multiplication. The cancellation effect is obvious in the latter case, where

(-1)*(-1) = 1.

The effect holds in the addition case, as shown below, but generally negative logs are not allowed in standard algebra.

ln-1 + ln-1 = ln(-1*-1) = ln(1) = 0

J provides control over the inverse operator because any inversion can be converted to a combination with J by the theorem of Inverse Promotion.

Inverse Promotion: <A> = ([A][<o>])

This theorem may be interpreted in addition space as

-a = a*(-1)

or in multiplicative space as

1/a = a^(-1)

Unlike the inverse boundary, J can be modified. For instance, giving J fractional cardinality makes it into a partial-inverse. In multiplicative space, the following half-cardinality of J is equivalent to i, the square root of -1:

Half-J: ([[<o>]]<[oo]>) = ([J]<[oo]>)

A pair of these makes a complete J. In multiplicative space, this translates to

i*i = -1

In additive space, its just

(ln-1)/2 + (ln-1)/2 = ln-1.

Complex numbers arise immediately from the Half-J. When space is multiplication, the complex number a+ib appears as:

[(a)(b ([J]<[oo]>))]

When space is addition, the Half-J is interpreted not as i but as the log of i, which equals i Pi/2. Thus J must be equal to i Pi. This interpretation coincides with Euler's formula:

Theorem: 1 + e^(i Pi) = 0

o ([<o>])	Given
o <o>	Involution
	Inverse

From J and Half-J, a radian value of Pi can be symbolically constructed using space as addition:

i Pi = J = [<o>]
i = e^(Half-J) = (([J]<[oo]>))

Pi = (-1) * i Pi * i
= ([<o>][J]([J]<[oo]>))
= (J [J] ([J]<[oo]>))

This construction of pi treats it not as a real number but as a phase construction that is irresolvable with other numbers. Its only semantic value comes out of the above construction, i.e. pi cannot be shown to equal 3.14159... using the semantic construct of boundary algebra.

The real values of e and pi are based on criteria independent of the algebra (e.g. geometries) and are not essential to the behavior of transcendental functions. In boundary algebra, their real values are truly unknown and therefore incommensurable with other quantities.

Thus, boundary algebra represents the basic transcendental values, summarized in Table 5 below. Using these values, transcendental functions can be constructed and evaluated.

Value	*Standard Form*	*Boundary Form*
J	ln-1	[<o>]
i	sqrt(-1)	(([J]<[oo]>))
pi	3.1415...	(J [J] ([J]<[oo]>))
e	2.7183...	(o)

Table 5. Transcendentals.

CONCLUSIONS

The mappings from traditional algebra to boundary algebra provide insight into standard algebra.

In boundary algebra, addition and multiplication play a secondary interpretation role aside from the primary concepts. Manipulations do not rely on these interpretations; instead they use constructs that are only vaguely present in standard algebra. Although the fundamental concepts of BA are found in addition and multiplication, these operations do not do justice to the simplicity and applicability of the concepts.

BA implements a generalized cardinality independent of addition or multiplication. The same cardinality theorems apply throughout various mathematical constructions because BA establishes a common basis for building functions and values. Counting is counting, regardless of the context.

BA implements a generalized inverse that is not bound to either addition or multiplication. Both functions utilize the inverse boundary to create inverted elements. The generalized inverse demonstrates that all inverses that it supports share certain properties: these properties can be derived independent of their functional context.

Boundary algebra reduces elementary algebra to a few concepts and a few syntactic mechanics, suggesting that it represents a more fundamental mathematical core than traditional forms. This reduction shifts the complexity away from basic mechanics into higher order structures. The resulting structures are larger but more insightful because the added elements reveal the dynamics of that structure: high level behaviors can be completely deduced from the low level mechanics and the low level mechanics are directly revealed in the visual forms.