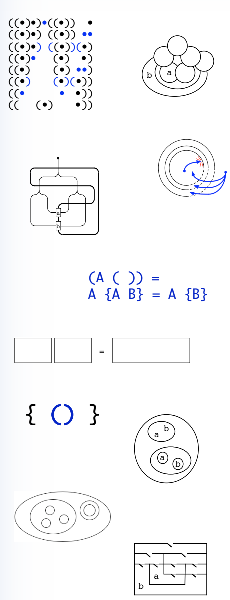
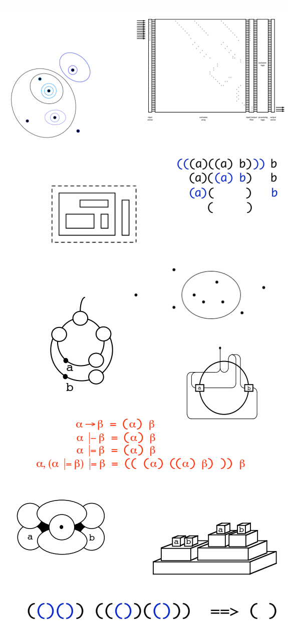
Syntactic Variety in Boundary Logic

A Presentation for Diagrams 2006

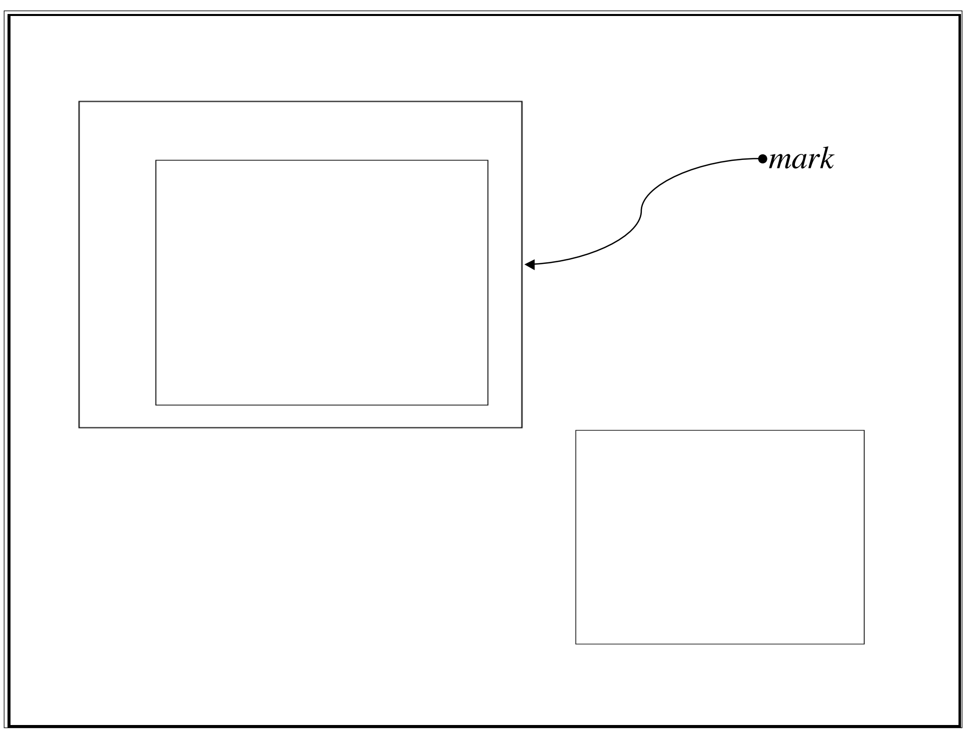
William Bricken
June 28, 2006
bricken@halcyon.com

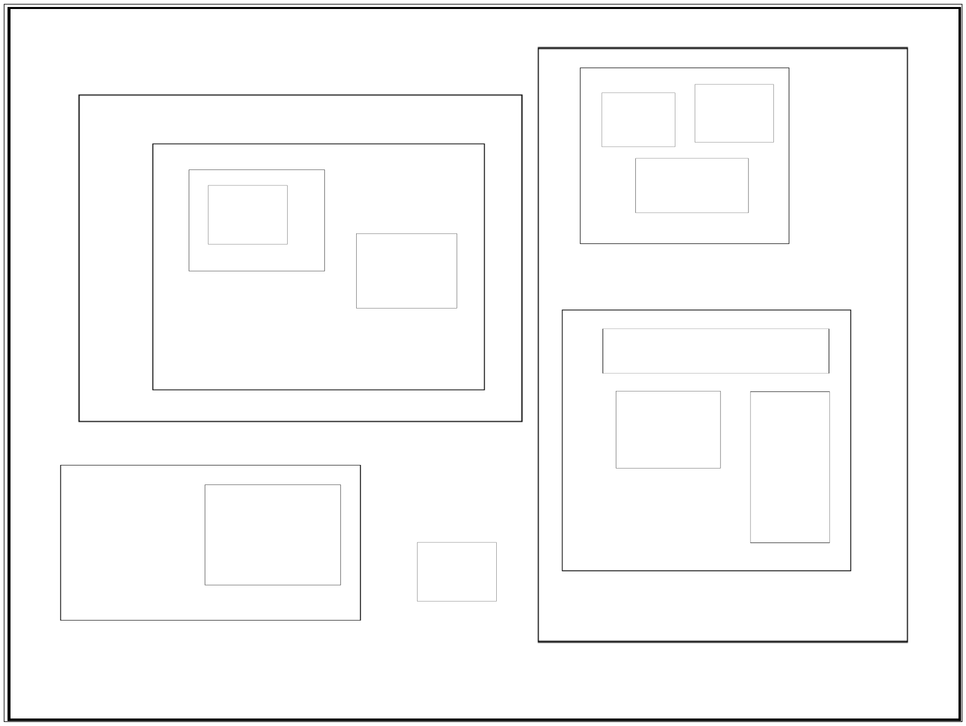
Contents





DE NOVO





Void and Mark

We construct a particular space of representation by framing it.

In the beginning there is no structure, there is only the frame.

Void space within the frame is featureless.
- void space is not filled with points, nor is it continuous

A frame is singular and cannot be decomposed.
- absence of decomposition means absence of the concept of intersection

A mark is a representation of the frame within itself.
- marks support both an inside and an outside (just like frames)

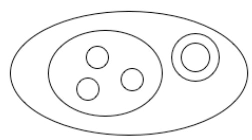
Replication of marks constructs a language of boundary forms.

Typographical Delimiters as Marks

A delimiting pair of tokens (parentheses, braces, brackets, quotations, etc.) can be used as a typographical representation of a mark.

Parentheses used as spatial boundaries are called *parens*.

```
( inside )   outside
          \
           parens boundary
```



= ((())(()()()))

Parens introduce these accidental properties (which must be ignored):

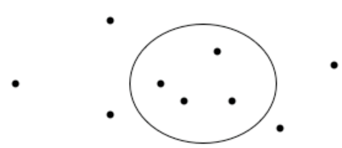
- fragmentation of the boundary into two tokens "left" and "right"
- linear ordering of boundaries in a string

Two Types of Composition

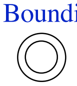
A boundary form is a composition of non-intersecting closed curves.

Boundaries support two types of composition

- SHARING is composition on the outside
- BOUNDING is composition on the inside
- neither operation requires a concept of arity

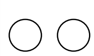


Bounding



Forms are bounded (contained or enclosed) by an outer boundary.
Any number of forms can be contained by the same boundary.

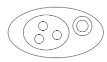
Sharing



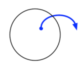
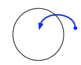
Forms sharing a space are structurally independent.
Any number of forms can share a space.
Forms within a common boundary share the interior space of that boundary.

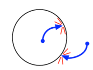
Two Types of Crossing

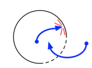
Boundaries impose structure on a featureless representational space by distinguishing their contents, nothing more.

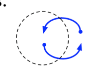


Boundaries have two sides, permitting two types of crossing:

from the inside to the outside  from the outside to the inside 

Impermeable boundaries deny crossing of both types. 

Semipermeable boundaries permit crossing in one direction only. 

Fully permeable boundaries do not distinguish their contents.
(They are indistinguishable from the absence of a boundary.) 

By convention, the semantic viewpoint is on the outside.*
(That is, *we* are on the outside of a representational space.)

* W. Bricken (1994) Inclusive Symbolic Environments. in K. Duncan and K. Krueger (eds.) *Proceedings of the 13th World Computer Congress*, v3, Elsevier Science, 163-170.

Pattern-Variables and Pattern-Equations

Pattern-variables stand in place of any form (i.e. universal quantification), including

- an outer boundary and its contents A = (()(()()))
- forms sharing a space A = (()()) (())
- the absence of a form A = <void>

Pattern-templates are forms (usually with variables) that identify an equivalence class.

Example: (A ((B))) matches ((()()) (())), with A=(()()) and B=<void>
as well as (((()())) ()), with A=() and B=()()
but not (() (()()))

Pattern-equations collapse specific equivalence classes.

- transformation of patterns is based on substitution and replacement of equals
- pattern-equations can be applied in parallel when matches do not overlap structurally
- pattern-equations define the semantics of the boundary language

Example: (A)(B) = (A B)

```
((())(()()))
((())(()()))    two parallel substitutions
(((  (   )))    one resultant sequential substitution
(((      )))
```

First Class Void

Empty containers permit the semantic use of non-representation.

( ) contains *nothing* on the inside

Void-equivalence (A ()) = <void>
- forms and pattern-templates can be equated to <void>.

Void-based pattern transformation (B (A ())) = (B)
- substitution of <void> for a void-equivalent form is deletion of the form

Void-substitution (B) = (A ()) (B (A () (A ())))
- void-equivalent forms can be deleted at will
- void-equivalent forms can be constructed anywhere throughout a form

~~~ The Principle of Void-Equivalence ~~~

Void-equivalent forms are syntactically irrelevant and semantically inert.

Two Interpretations

A (semantic) interpretation is a mapping of boundary forms to objects in a domain of interest, together with pattern-equations that specify the calculus of the domain.

Many interpretations of boundary mathematics have been developed, including knot theory, Boolean algebra, real numbers, and imaginary logic and numerics.*

Two interpretations follow, integer arithmetic and propositional logic.

Object mapping:	0 = <void>, 1 = ()	FALSE = <void>, TRUE = ()
Corresponding operations:	Addition is SHARING Multiplication is BOUNDING	Disjunction is SHARING Negation is BOUNDING
Pattern-equations:	(())=()()	()=()(), (())=<void>

The syntactic varieties presented later apply to any interpretation.

* W. Bricken (1991) A Formal Foundation for Cyberspace. *Proceedings of Virtual Reality '91, The Second Annual Conference on Virtual Reality, Artificial Reality, and Cyberspace*, San Francisco, Meckler, 9-37
* W. Winn and W. Bricken (1992) Designing Virtual Worlds for Use in Mathematics Education: The Example of Experiential Algebra. *Educational Technology*, v32(12), 12-19.
~~~

BOUNDARY INTEGERS

Boundary Integer Arithmetic*, Representation

Boundary place notation

uses depth of nesting rather than location in sequence for place notation.

( ) is atomic
and cannot
be decomposed

```
0          <void>
1       •  = ( )
2      ••  = (•)
3     •••  = (•)•
4    ••••  = (•)(•) = (••) = ((•))
```

stroke arithmetic   merge   double

Pattern-equations for standardizing forms:

•• = (•)	**Double**
(A)(B) = (A B)	**Merge**

Standardization constructs an equivalent form with the fewest number of boundaries.

*Kauffman, L.H. (1995) Arithmetic in the Form. *Cybernetics and Systems* 26: 1-57.

Boundary Integer Arithmetic, Operations

Addition is sharing the same space:

```
A + B   ==>   A B
```

Multiplication is unit substitution:

```
A * B  ==>  substitute[B for • in A]  =  substitute[A for • in B]
```

- Addition occurs by placing forms in the same void-space
  - no ordering, grouping, or arity in void-space
- Multiplication occurs by placing replicate forms in •-space
  - no ordering, grouping, or arity in •-space
- Neither operation requires additional computation.
  - no number facts, no "carrying"

All computation is form standardization.

Boundary Integer Operations, Example

```
5: ((•))•          7: ((•)•)•          7+5: ((•)•)•  ((•))•
```

```
5*7:  ((   •    ))    •     *  ((   •    )   •    )   •
      ((   7    ))    7     =  ((   5    )   5    )   5
      (( ((•)•)• )) ((•)•)• =  (( ((•))• ) ((•))• ) ((•))•
```

Standardizing the results: (D = double M = merge L = linear artifact)

7+5: put 7 and 5 in space

```
((•)•)•((•))   •    L
((•)•) ((•)) ••     D
((•)•) ((•))(•)     M
((•)•    (•)   •)   L
((•)     (•) ••)    D
((•)     (•)(•))    M
((•         •  •))  D
((    (•)      •))
```

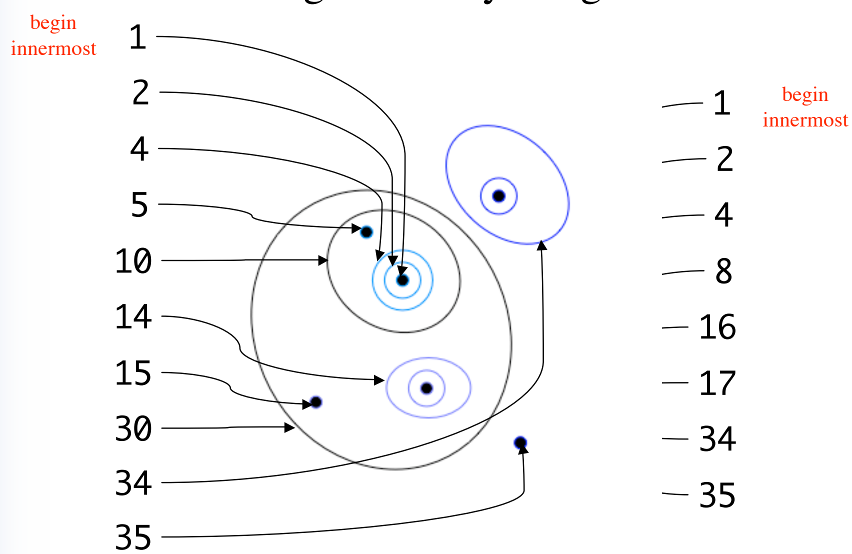
5*7: substitute 7 for • in 5

```
((((•)•)•))((•)•)•    M
((((•)•)•)  (•)•)•    M
((((•)•)•    •)•)•    D
((((•)•)  (•) )•)•    M
((((•)•     •) )•)•   D
((((•)  (•) ) )•)•    M
((((•     •) ) )•)•   D
((((  (•) ) ) )•)•
```

7*5: substitute 5 for • in 7

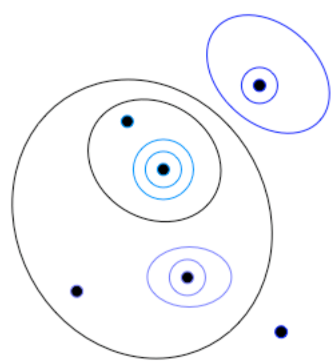
```
((((•))•)((•) )•)(( •) )•    M
((((•))•  (•) )•  ( •) )•    L
((((•))   (•)•)   ( •)•)•    M
((((•)      •)•     •)•)•    D
((((•)      •)   (•)  )•)•   M
((((•)      •     •)  )•)•   D
((((•)         (•)  )  )•)•  M
((((•           •)  )  )•)•  D
((((      (•)      )  )  )•)•
```

Reading Boundary Integers



Standardizing Boundary Integers

Standardization reduces the boundary form of 35 from 14 to 8 boundaries



BOUNDARY LOGIC

The Evolution of Boundary Logic

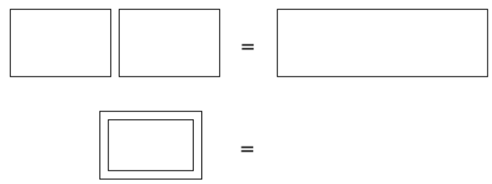
Originated by the *founders of formal logic*

- 1879 Gottlob Frege -- network notation based on implication
  the German logician who invented formal mathematics
- 1896 Charles S. Peirce -- enclosure notation based on conjunction
  the American logician who invented semiotics

1890s C.S. Peirce	entitative and existential graphs, boundary notation
1963 I. Calvino	"A Sign in Space" (literature)
1967 G. Spencer Brown	"Laws of Form" (mathematics)
1975 F. Varela (and L. Kauffman)	"A Calculus for Self-reference" (imaginaries)
1982 W. Bricken	Losp Deductive Engine (computer science)
1985 First Sign/Space Conference	(cybernetics)
1992 R. Shoup	"A Complex Logic for Computation with Simple Interpretations for Physics" (physics)
2001 L. Kauffman	"The Mathematics of Charles Sanders Peirce" (mathematics)

A Boundary Arithmetic for Logic

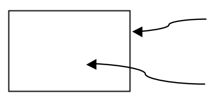
Common boundaries cancel *(when empty on the inside)*



Call

Cross

Interpretation



The outside is TRUE. T = $\boxed{}$

The inside is FALSE. F = <void>

One spatial form provides two symbolic values.

Boundary Logic Pattern-Equations

SHARING EVALUATION *(idempotency)*

( )( ) = ( ) **Call** composition on the outside is disjunction

BOUNDING EVALUATION *(involution)*

(( )) = <void> **Cross** composition on the inside is negation

Spencer Brown's radical innovation

Each pattern-equation is implemented by pattern-matching and substitution.
Each proceeds from left to right by void-substitution.

Evaluation via Deletion

FALSE = 0 = <void>

To evaluate a FALSE variable: *delete* the variable.

TRUE = 1 = ( )

To evaluate a TRUE variable: *delete the container* of the variable.

Example:

a IFF b -->	(a b) ((a)(b))		transcribe
Let a=0, b=0:	( ) (( )( ))	==> ( )	call, cross
Let a=0, b=1:	(()) (( )(()))	==> <void>	cross 3 times
Let a=1, b=1:	(()()) ((())(()))	==> ( )	call, cross 3 times

Transcribing Boolean and Boundary Logics

BOOLEAN	BOUNDARY
FALSE	<void>
TRUE	( )
NOT a	(a)
a OR b	a b
NOT (a OR b)	(a b)
IF a THEN b	(a) b
a AND b	((a)(b))
a EQUIVALENT b	(a b)((a)(b))

The boundary logic "constant" set: { () }
The boundary logic "function" set: { () }

One-to-Many Mapping

One boundary form represents *many* different conventional logic expressions.

A one-to-many mapping is necessary for one system to be *simpler*.

The particular logical interpretation of a given boundary form is a *free choice*.*

```
( )

1
NOT 0
1 OR 0
0 OR 1 OR 0
0 NOR 0
(NOT 0) OR 0
NOT (0 OR 0)
NOT (0 OR 0) OR (0 OR 0)
...
```

```
((a)(b))

a AND b
b AND a
NOT (NOT a OR NOT b)
NOT a NOR NOT b
NOT (a NAND b)
(a AND b) OR 0
NOT (a NAND (0 OR b)) OR 0
NOT (b NOR 0) OR NOT a OR 0
...
```

<void> = 0 = 0 OR 0 = 0 OR 0 OR 0 = ...

*Shin, S. (2002) *The Iconic Logic of Peirce's Graphs*. MIT Press

Algebraic Pattern-Equations

Axioms

remarkably succinct

(A ()) = <void> **Occlusion**

A {A B} = A {B} **Pervasion**

Curly braces refer to *any* deeper intervening structure.
There is no analogy in conventional mathematical techniques.

Useful Theorems

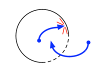
((A)) = A **Involution**

() A = () **Dominion**

Each pattern-equation is implemented by pattern-matching and substitution.
Each proceeds from left to right by void-substitution.

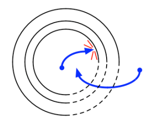
Deep Transformation

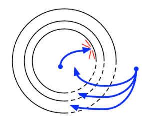
Any form on the outside of a boundary pervades all inside spaces.
From the outside, boundaries are *transparent* (semipermeable).



`A {A B} = A {B}` **Pervasion**

Forms in an exterior space are arbitrarily present in every interior space.





Example:

```
a (b (a c (d (a b e))))
a (b (  c (d (  b e))))        pervasion a
a (b (  c (d (    e))))        pervasion b
```

Therefore: `a (b (a c (d (a b e)))) = a (b (c (d (e))))`

Boundary Logic Proof of *Modus Ponens*

α, (α |= β) |= β --> (((α) ((α) β))) β

Transcribe

```
 (a AND (a IMPLIES b)) IMPLIES b          modus ponens
 (a AND (a IMPLIES b)) IMPLIES b          a IMPLIES b --> (a) b
 (a AND    (a) b     ) IMPLIES b          a AND X --> ((a)(X))
 ((a)   (   (a) b  )  ) IMPLIES b         X IMPLIES b --> (X) b
( ((a)   (   (a) b  )  ) )    b
```

Reduce

```
(((a)((a) b))) b         transcription
 (a)((a) b)    b         involution    ((A)) ==> A
 (a)(     )    b         pervasion     A (A B) ==> A (B)
    (     )              dominion      A ( ) ==> ( )
```

Interpret TRUE

Pathsways Through Metatheory

Void-equivalence

《(A ())》 = (《A ()》) = (《A》 《()》) = (《A》 ()) = (()) = <void>

The map to logic

Metatheory is invariant under provable equivalence.

- the maps from Boolean logic and from AEG to boundary logic preserve validity*

Algebraic deduction

Entailment transcribes into Birkoff's rules of equational deduction**.

- validity is maintained by bidirectional equations
- substitution and replacement are domain independent

```
A=B IF  A⟹B
A=B IF  A⟹C  AND B⟹C
A=B IF  A⟹() AND B⟹()
     OR A⟹   AND B⟹
A=B IMPLIES  φ[A]=φ[B]
```

Algebraic structure

Completeness follows from the maximal ideal theory***.

Pattern rewrite system

Void-substitution assures both termination and convergence.

Induction over boundary patterns

Ground cases: () and <void>.
Given (A), show A
Given A B, show A , B separately

Net result:

$\alpha \rightarrow \beta$ = (α) β
$\alpha \vdash \beta$ = (α) β
$\alpha \models \beta$ = (α) β
α, ($\alpha \models \beta$) $\models \beta$ = (((α) ((α) β))) β

*Dau, F. (2005) *Mathematical Logic with Diagrams.* www.dr-dau.net/publications.shtml
**Birkoff, G. (1935) On the Structure of Abstract Algebras. *Proceedings of the Cambridge Philosophical Society*, 31 417-429
***Halmos, P. and Givant, S. (1998) *Logic as Algebra.* Mathematical Association of America.

Alpha Existential Graphs

Peirce's five rules for Alpha Graphs* map directly to boundary logic pattern-equations:

RULE	EXISTENTIAL	ENTITATIVE	BOUNDARY
R1. Erase	((A) B) |= (() B)	((A) B) |= ((A))	} (A ()) =
R2. Insert	(A) |= (A B)	A |= A B	
R3. Iterate	A (B) |= A (A B)	A (B) |= A (A B)	} A {B} = A {A B}
R4. Deiterate	A (A B) |= A (B)	A (A B) |= A (B)	
R5. Double cut	((A)) = A	((A)) = A	((A)) = A

Boundary logic provides a more modern algebraic transformation system, uniting pairs of asymmetrical implicative rules into symmetrical equations.

The Erase and Insert rules of AEG fail to provide a clear termination goal. Boundary logic uses a single void-equivalence rule, **Occlusion**, as a termination condition.

* Peirce, C.S. (1931-58) *Collected Papers of Charles Sanders Peirce.* Hartshorne, C. Weiss, P., Burks, A. (eds.) Harvard Univ Press.

SYNTACTIC VARIETY: REPRESENTATION

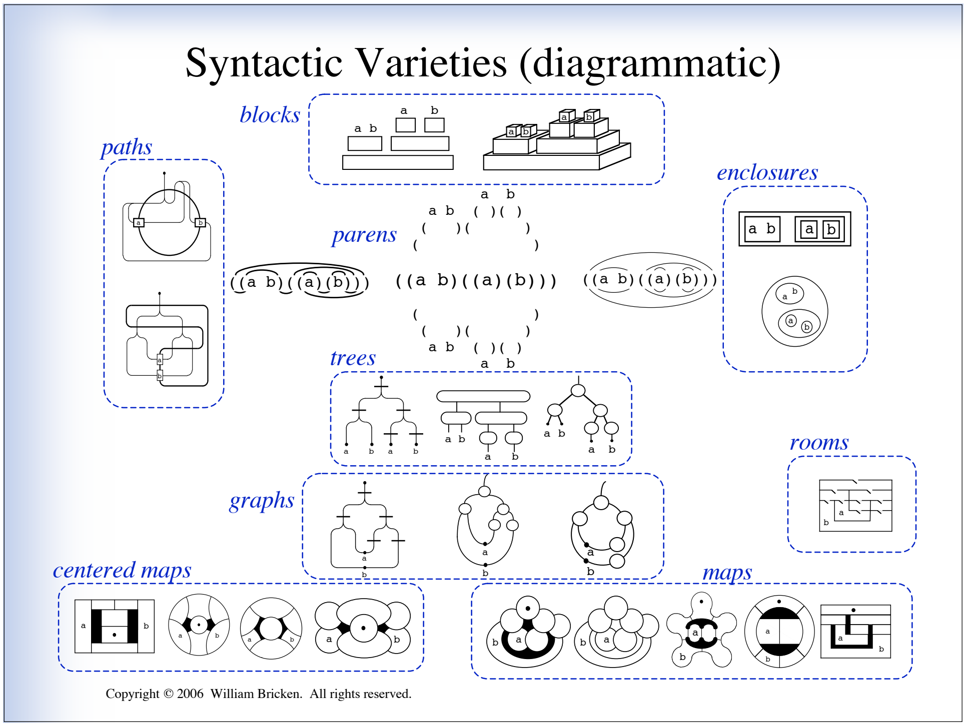
Syntactic Varieties (textual)

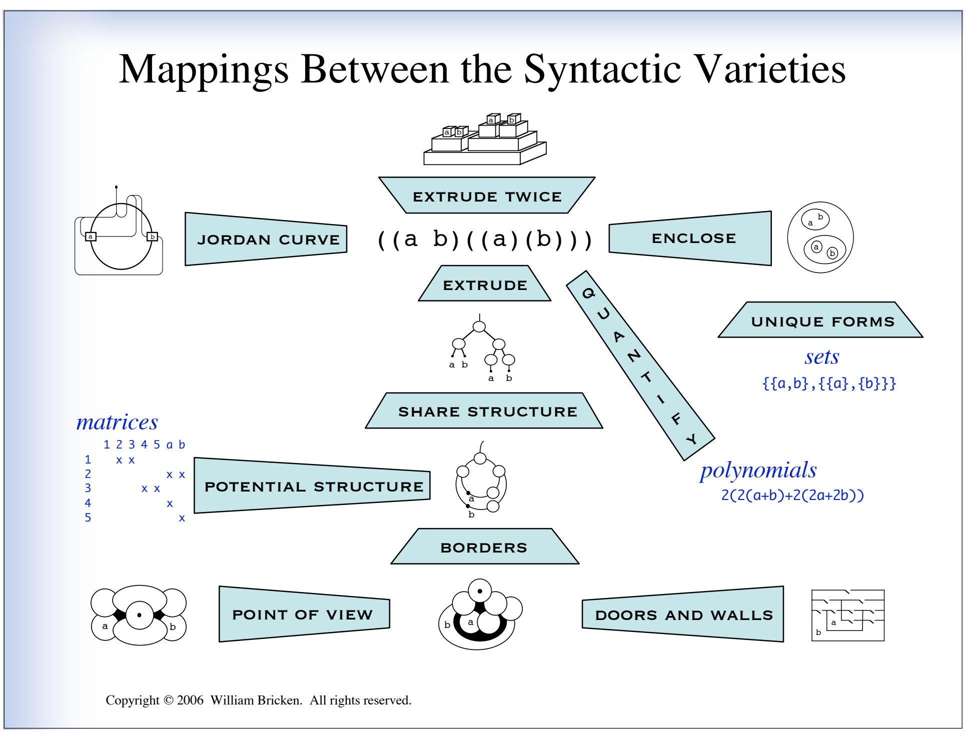
Each syntactic variety that follows assumes **Occlusion** and **Pervasion**, the two pattern-equations that define boundary *logic*.

Topological varieties
- different spatial data structures with different implementation behavior
- analogous to conventional data structures

Geometric varieties
- different metric structures with similar implementation behavior
- analogous to exchanging tokens in a string-based language

VARIETY	DIMENSION	BOUNDING	SHARING	POINT OF VIEW
parens	1	nest	space	outside
enclosures	2	enclose	space	outside
trees	2	link	branch	outside
maps	2	border	common neighbor	outside
centered maps	2	border	common neighbor	inside
rooms	2/3	door	common neighbor	inside
graphs/networks	3	link	branch	both
paths	3	cross	fork	both
blocks	3	stack	common floor	outside


Syntactic Varieties (diagrammatic)
blocks
paths
enclosures
parens
((a b)((a)(b)))
trees
rooms
graphs
centered maps
maps
Copyright © 2006 William Bricken. All rights reserved.


Mappings Between the Syntactic Varieties
EXTRUDE TWICE
JORDAN CURVE
((a b)((a)(b)))
ENCLOSE
EXTRUDE
QUANTIFY
UNIQUE FORMS
sets
{{a,b},{{a},{b}}}
SHARE STRUCTURE
matrices
POTENTIAL STRUCTURE
polynomials
2(2(a+b)+2(2a+2b))
BORDERS
POINT OF VIEW
DOORS AND WALLS
Copyright © 2006 William Bricken. All rights reserved.

Syntactic Concepts

Dimensionality of representation
1-space fractures containment, 2-space limits structure sharing

Top and bottom
represents outermost and innermost

Point of view
read from outside (objectively) or from inside (subjectively)

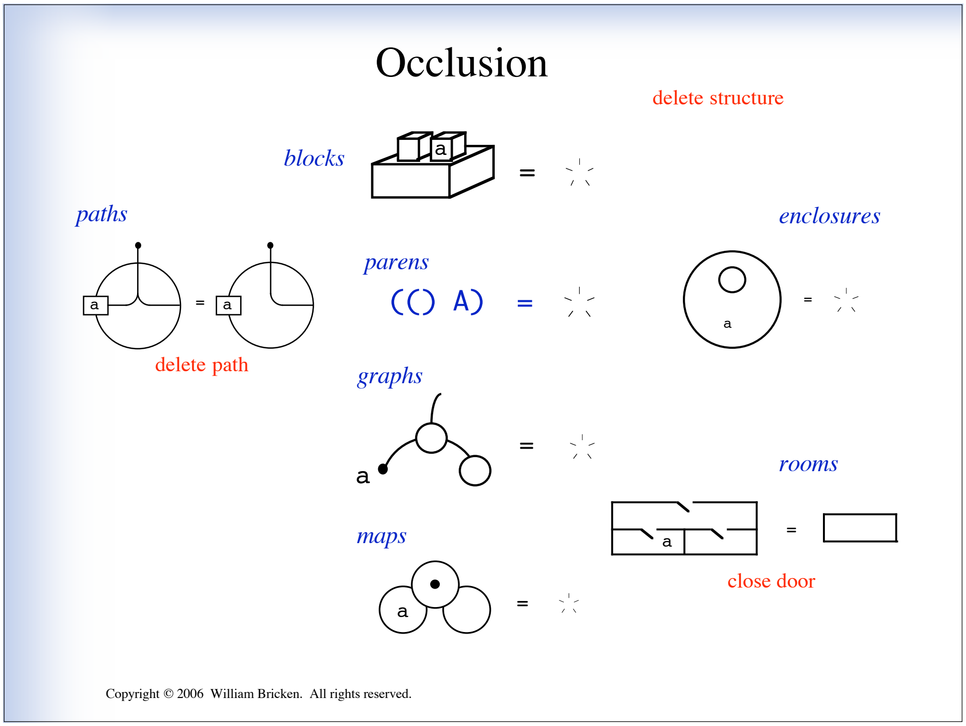
Anthropomorphism
some forms are physically familiar, others are abstract

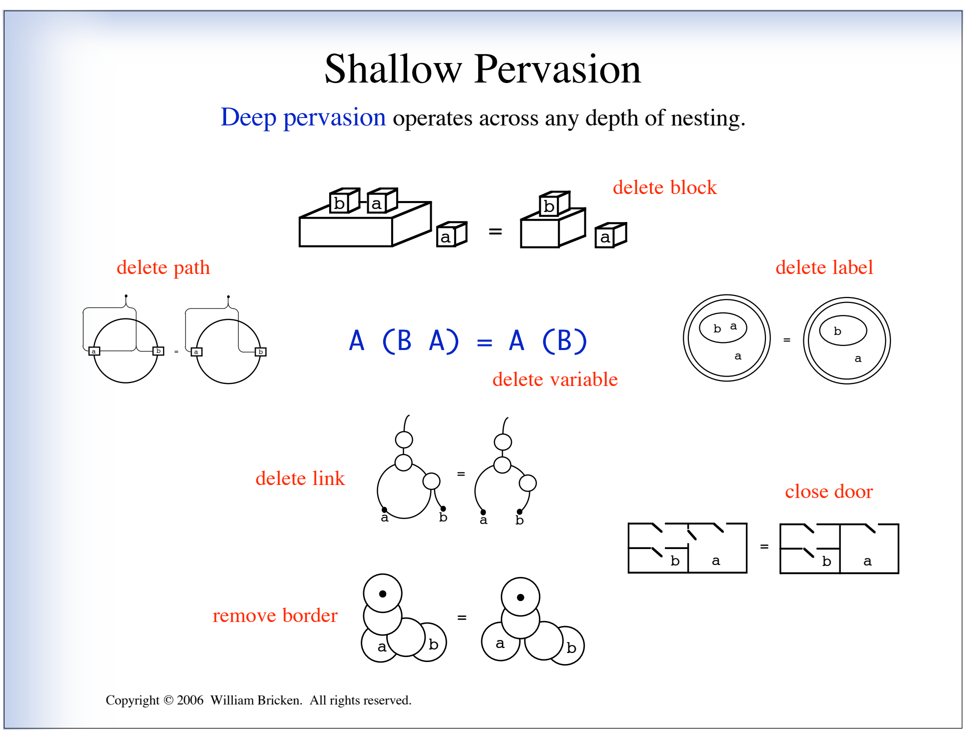
Surrounding space
map varieties incorporate the background substrate

Geometric varieties
rubber sheet geometry

Topological varieties, generated by
extrude and rotate in higher dimensional space
structure sharing (unique objects)
convert links to borders
exterior or interior point of view
exchange objects for processes

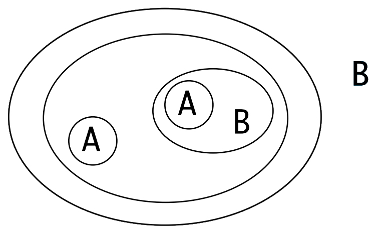
SYNTACTIC VARIETY: TRANSFORMATION


Occlusion
delete structure
blocks
paths
enclosures
parens
(() A) =
delete path
graphs
rooms
maps
close door
Copyright © 2006 William Bricken. All rights reserved.


Shallow Pervasion
Deep pervasion operates across any depth of nesting.
delete block
delete path
delete label
A (B A) = A (B)
delete variable
delete link
close door
remove border
Copyright © 2006 William Bricken. All rights reserved.

Modus Ponens: Parens and Enclosures

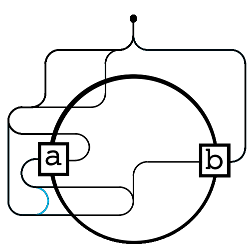
```
( ((A) ((A) B)) ) B = ()
```

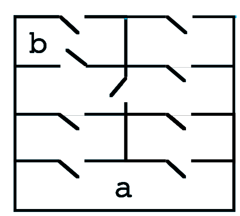


interpret as TRUE

```
(((A) ((A) B))) B     transcription
(((A) (     ))) B     pervasion (A) B
(             ) B     occlusion
(             )       dominion
```

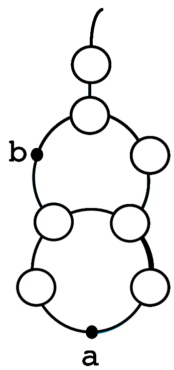

Modus Ponens: Paths and Rooms






```
(((a) ((a) b)))   b     transcription
 (a) ((a) b)      b     involution
 (a) ((a)  )      b     pervasion  b
 (a)   a          b     involution
 ( )   a          b     pervasion  a
 ( )                    dominion
```

Modus Ponens: Distinction Networks



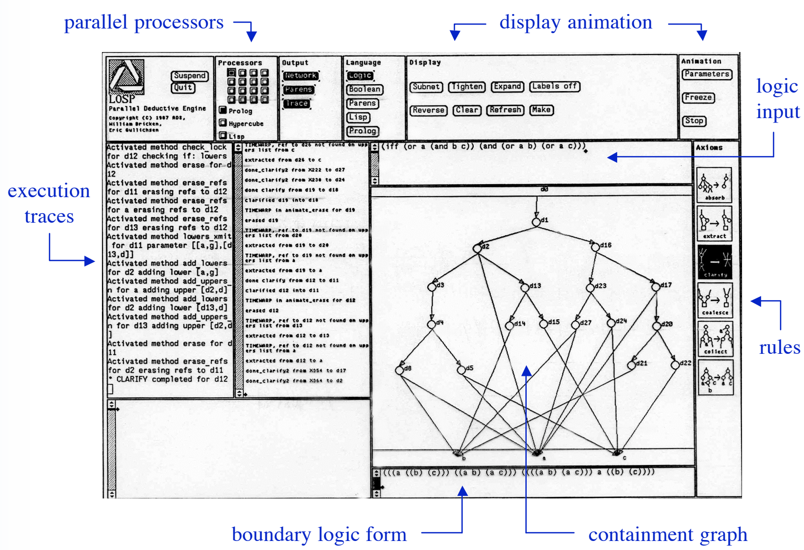

```
(( (((a) ((a) b)))  b ))      transcription
(( (((a) ((a)   )))  b ))     pervasion  b
(( (((a)   a     ))  b ))     involution
(( ((( )   a     ))  b ))     pervasion  a
(( (             )  b ))      occlusion
(                      )      occlusion
```

Concurrent, asynchronous network reduction*

- each node is an atomic autonomous agent
- communication with direct neighbors only
- local interaction leads to
  global deduction without global coordination

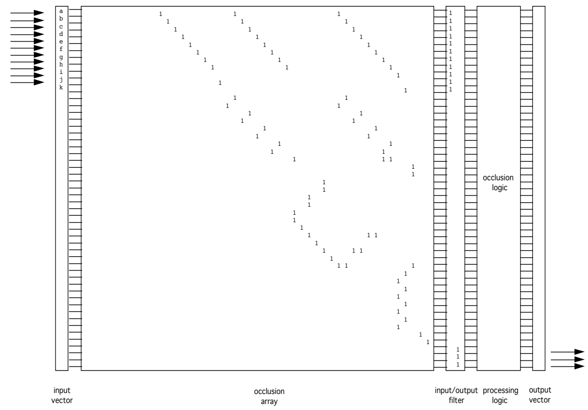
*W. Bricken (1995) Distinction Networks. in I. Wachsmuth, C.R. Rollinger & W. Brauer (eds.) *KI-95: Advances in Artificial Intelligence.*, Springer, 35-48.

The Losp Parallel Deduction Engine* (1987)



* W. Bricken and E. Gullichsen (1989) An Introduction to Boundary Logic with the Losp Deductive Engine, *Future Computing Systems* 2(4), 1-77.

Sparse Containment Matrix (2002)



Distinction networks implemented on a reconfigurable silicon hardware substrate.
Matrix entries route signals locally.
Signals traversing the containment matrix implement logic network functionality.

In Conclusion

Boundary logic is an efficient, scalable, robust diagrammatic logic based on pattern-substitution.

Boundary integers exchange the effort of addition and multiplication for a single standardization process.

Both are interpretations of the same simple diagrammatic language of non-intersecting spatial enclosures.

Boundary languages have unique syntactic varieties generated by topological and geometric transformation of structure.

This presentation is available at `www.wbricken.com/01bm/0103notate`

I'll be available throughout the conference to demonstrate an implementation of boundary logic used for minimization of commercial semiconductor circuits.

Thank you!

Comments and questions gladly accepted.
bricken@halcyon.com

SUMMARIES

Presentation Notes

Boundary mathematics is a *fundamental innovation in mathematics* (!).

The entertaining challenge:
- do not force-fit these ideas into pre-existing conceptual structures
- very easy to understand on its own ground
- somewhat difficult to understand using conventional concepts

These mathematical techniques have been extensively tested.
- implemented in literally dozens of programming languages
- applied to SAT problems, theorem proving, expert systems
- applied to industrial strength problems in semiconductor minimization

The presentation style is unorthodox.
- rapid visual exposure to relatively dense information
- seeds for contemplation rather than an immediate explanation

Some slides are included for completeness, and will not be discussed in depth.

The presentation (and lots of other material on Boundary Math) is available at
`www.wbricken.com/01bm/0103notate`

Non-Conventional Mathematics Warning

If a concept or a representation is not explicitly permitted, it is forbidden.

Void space means with no pre-assumed mathematical or structural concepts.
- the space of representation is unstructured
- drawing a mark introduces a distinction; it does not introduce a topology (no points) or a geometry (no metric)

In specific, absence of the concept of arity implies
- absence of the capability to count
- no conventional functions or relations
- associativity and commutativity are not relevant structural concepts
- the inside/outside distinction made by boundaries is not relational (boundaries are not set objects)

In general, these mathematical concepts have not been explicitly introduced:

– sets	{a,b}
– points	(a,b)
– counting and arity	1,2,3,…
– functions and relations	f(a) r(a,b)
– logic	a AND b
– group theory	$a \lozenge a^{-1} = i$
– categories	$f(a) \lozenge f(b) = f(a \lozenge b)$

Boundary Mathematics

Representation

- Forms: a language of configurations of closed non-overlapping curves

  ◯ is a form
  If A is a form, so is Ⓐ
  If A and B are forms, so is A B

- Variables: tokens standing in place of arbitrary forms
- Patterns: forms serve as structural templates to identify members of an equivalence class
- Pattern-equations: pairs of patterns that collapse equivalence classes

Transformation

- defined solely by pattern-equations
- substitution and replacement of equivalent patterns

Strategy

- extreme minimalist
- begin on entirely new conceptual ground
- semantic use of void-space (forms can be void-equivalent)

Novel Concepts

Semantic void/single constant
Void is everywhere; boundaries distinguish their contents. Nothing more.

Inside and outside of forms
Enclosure has an interpretation as a partial order.

Void-equivalence
Void-equivalent structure is semantically irrelevant and semantically inert.

Semipermeable boundaries/operational transparency
Boundaries are barriers to their contents, but can be transparent to their context.

Object/operator unification
Patterns are both objects and operators.

Spatial (non-linear) notation
Inherent computational parallelism.
Syntactic varieties are generated from spatial transformation of forms.

Why Isn't Boundary Logic Better Known?

Avoidance of the void
- "concepts must have symbolic representations"
- non-existence cannot contribute to computation
- separate concepts must have unique representations
- computation occurs in discrete, unambiguous steps

0110101100
_11_1_11__
11 1 11

The politics of symbolic mathematics
- "diagrams cannot be computational objects"
- Cartesian duality (17th century)
- Russell and Whitehead, *Principia Mathematica* (1910)

The danger of eccentrics
- "just another Boolean algebra" (the isomorphism critique)
- "the foundations of mathematics are well understood"

Many misconceptions and misinterpretations
- representing <void> with a token
- assuming a relational structure
- some trade secrecy

<void> = { } = Φ

a b = a R b

Boundary Logic Computation

An algebraic system
- primary semantics is *equality*
- primary process is *pattern-matching and substitution*
- axiomatized by two simple pattern-equations

A single concept system
- the primary object is the *boundary*
- the only structure is *enclosure (inclusion)*
- maps *one-to-many* onto Boolean techniques

A spatial system
- ordering, grouping and arity are not concepts within the system
- transformations within a space are in *parallel*

A void-based system
- *deletion* (void-substitution) rather than rearrangement
- boundaries are *transparent* from the outside
- forms sharing a space are *independent*

Algebra of Boundary Constants -- Metatheory

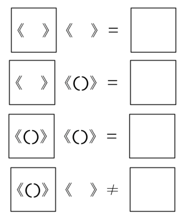
Void-equivalence is unorthodox.

no proofs here

For all boundary forms,
show that mark-equivalence, 《()》, and void-equivalence, 《 》,
never intermix during fully reductive pattern-substitution.

《()》 = (《》)

《()》 ≠ 《 》



《 》 = <void>

[] 《()》 = [] **Call**

A 《()》 = () **Call**

[《()》] = <void> **Cross**

sound and consistent

Cross and **Call** are the constructors of the language.

(《 》) ↔ A () ↔ A (A) ↔ A (A 《 》) 《 》 ↔ ...

《 》 ↔ (()) ↔ (A ()) ↔ (A (A)) ↔ ...

Different Interpretations of the Same Language

The semantics, or interpretation, of a boundary form is determined by the set of pattern-equations taken to be axiomatic.

Logic, under **Occlusion** and **Pervasion**

(((a)((a) b))) b ==> ()

interpret () as TRUE

Integers, under **Double** and **Merge**

(((a)((a) b))) b ==> (((a (a) b))) b

interpret as 2*2*2(a+2a+b) + b = 24a + 9b

For Contemplation

New mathematical systems, particularly for logic,
question our understanding of rationality, and
tend to question our understanding of reality.

Do syntactic diagrammatic varieties suggest new cognitive techniques?

Which aspects of our mathematical knowledge are purely historical,
and which are fundamental? (eg why is diagrammatic logic not preferred?)

Are algebraic concepts such as commutativity and transitivity
fundamental to cognition, or could they be artifacts of notation?

What is the interaction between syntax (ie data structure) and learning?

Do specific representations have affordances for errors?

What do void-equivalence and semipermeable boundaries have to do
with the logic embedded in language (other than functional equivalence)?
. . .

DIAGRAMMATIC REPRESENTATION

Wanted: A Theory of Representation

Variation in syntax is not addressed by mathematical morphism.

In Computer Science, data structures and algorithms that implement isomorphic structures vary profoundly, in succinctness, efficiency and understandability.

In Math Education, meaning is ignored in favor of manipulation of representations.

Semantic density (the amount of information carried by a representation) changes qualitatively with the dimension of a representation.

"house"    

Representation alone can introduce new concepts. For example, the expression "4 - 7" is either invalid, or requires an extension of the positive integers.

Operations are not independent of representation. How a positive integer is represented determines how addition and multiplication are performed.

Read/Operate Tradeoff: Positive Integers

System	Example	Read	Standardize	Add	Multiply
stroke	///// ///// ///// //	*very hard* count result	*trivial* unique integers	*trivial* push together	*easy* substitute replicate for each stroke
Roman	XVII	*moderate* add groups	*moderate* collect, promote groups	*trivial* push together	*very hard* compound rules
decimal	17	*easy* place notation	*easy* fixed places, decimal point	*hard* 100 facts, carry	*hard* 100 facts, accumulate
binary	10001	*easy* place notation	*trivial* fixed places	*moderate* 4 facts, carry	*moderate* 4 facts, accumulate
boundary	((((•))))•	*easy* depth notation	*moderate* double, merge	*trivial* push together	*easy* substitute replicate for each •

Representational Spaces

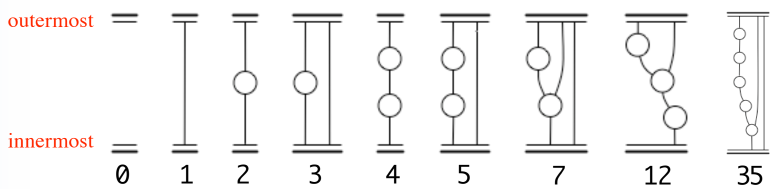
A representational space *is that space set aside by a boundary between physicality and virtuality.*

We are surrounded by instances of representational space:

a page
the blackboard
the projector screen
the frame of a painting
a transmission line
this slide
the television screen
the computer monitor
a book
where our voices are when we use a telephone

BOUNDARY INTEGERS: GRAPH VARIETY

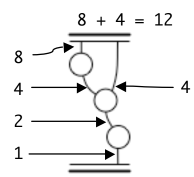
Boundary Integers, Network Variety

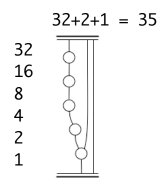


To read the network variety of boundary integers:

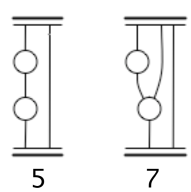
- follow each path from bottom-to-top
- for each path, begin with 1 at the bottom
- double the current result when passing through a node
- add paths together at the top

This is the same procedure used to read binary numbers.

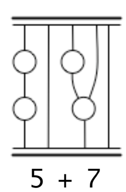




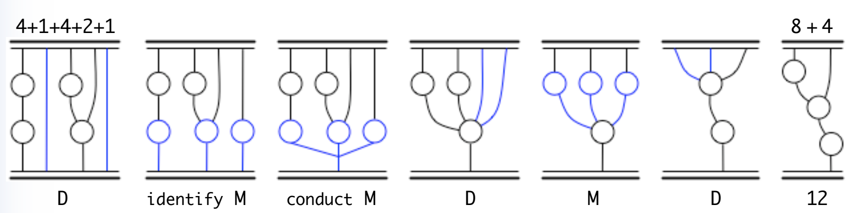
Boundary Integers, Network Add



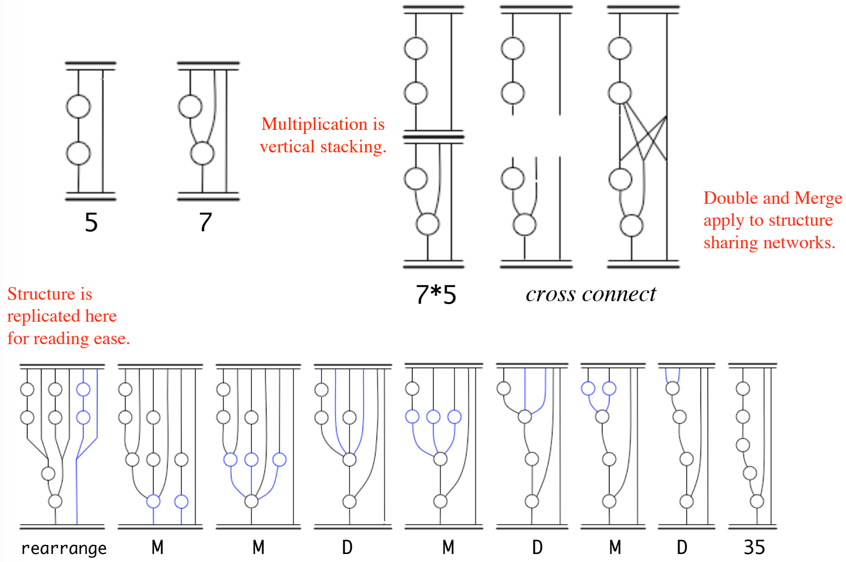
Addition is horizontal stacking.



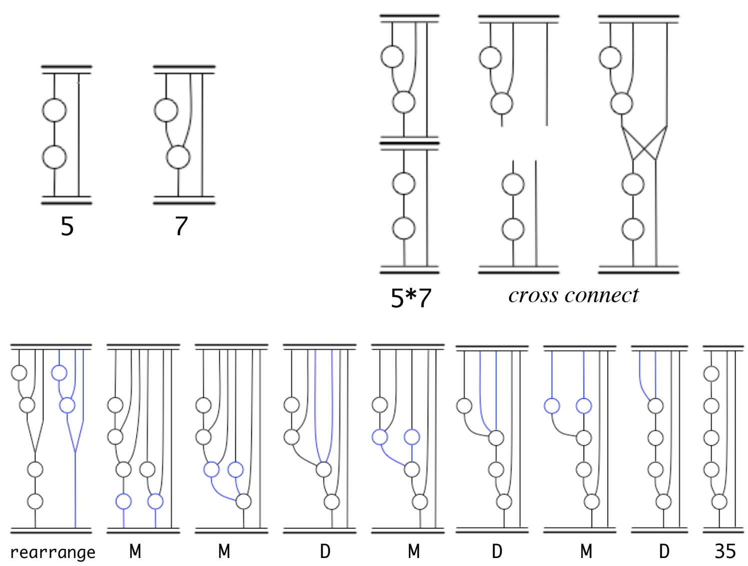
Standardizing the result:



Boundary Integers, Network Multiply I



Boundary Integers, Network Multiply II



BOUNDARY LOGIC: SUPPORT MATERIAL

Reading Mark as Implication

In implicative logic, valid implications maintain the truth value of an expression.
In algebraic logic, valid substitutions maintain the truth value of an equation.
In boundary logic, void-substitutions cannot change the truth value of a form.

Boolean	Boundary	
FALSE IMPLIES FALSE = TRUE	[ ] = ( )	identity
FALSE IMPLIES TRUE = TRUE	[ ] ( ) = ( )	call
TRUE IMPLIES FALSE = FALSE	[( )] =	cross
TRUE IMPLIES TRUE = TRUE	[( )] ( ) = ( )	cross

Several Proofs of a Simple Theorem

```
A ( ) =?= ( )
```

Direct transformation:

```
      A ( )                   lhs
    (( A ( ) ))               involution      A ==> ((A))
    (         )               occlusion, rhs  (A ()) ==>
```

Mutual transformation:

```
   A ( )  =?=  ( )
  (A ( )) =?= (( ))           F[A] = F[B]
           =                  occlusion twice, identity
```

Case analysis:

```
  () ( )  =?=  ( )            subst[(),A,E], arithmetic
     ( )  =?=  ( )            subst[  ,A,E], identity
```

Standard form:

```
  (A () ()) ((A ()) (()))     (A B)((A)(B))
            (           )     occlusion, 3 times
```

```
A ( ) = ( )
```

Dominion

Deep Transformation Example

Minimize: `((NOT b) OR NOT(a OR (NOT c))) AND ((a AND b AND c) OR NOT(a OR b))`

`(¬b ∨ ¬(a ∨ ¬c))  ∧  ((a ∧ b ∧ c) ∨ ¬(a ∨ b))`

Transcribe: `( ((b)(a (c))) (((a)(b)(c))(a b)) )`

Boundary Reduction:

```
    ( ((b)(a (c))) ((             (a)(b)(c))(a b)) )   eg
1   ( ((b)(a (c))) ((((b)(a (c)))(a)(b)(c))(a b)) )   per+
2   ( ((b)(a (c))) (((    (a (c)))(a)(b)(c))(a b)) )   per (b)
3   ( ((b)(a (c))) ((      a (c)  (a)(b)(c))(a b)) )   inv
4   ( ((b)(a (c))) ((      a (c)  ( )(b)(c))(a b)) )   per a
5   ( ((b)(a (c))) (                        (a b)) )   occ
6   ( ((b)(a (c)))                           a b   )   inv
    ( (( )(a (c)))                           a b   )   per b
    (                                        a b   )   occ
                        ¬(a ∨ b)                      interpret
```

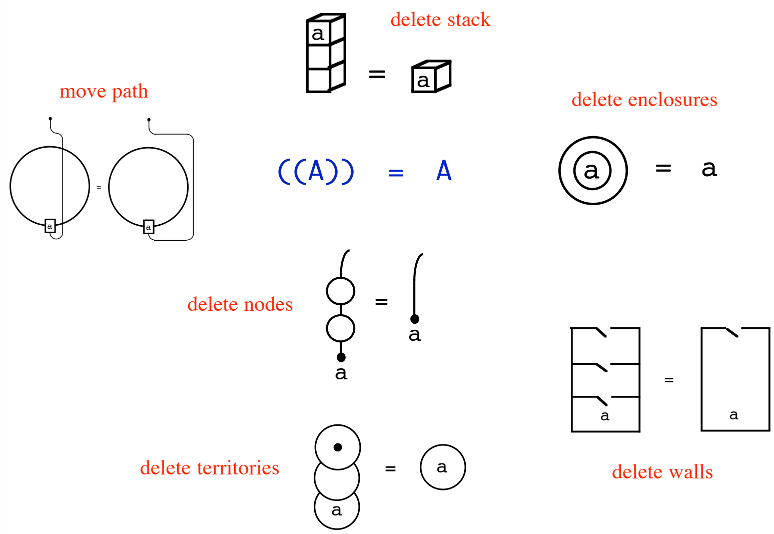
What if boundaries were interpreted as functions on their contents?

1. A compound function is added as an argument to an external boundary function.
2. An argument to the compound function is deleted, changing its arity.

3 and 5. Functional inverses created by the deleted argument cancel, creating two new simple arguments.

4. One of the simple arguments voids its containing function.
6. One of the simple arguments voids the original compound function by voiding one of *its* arguments.

Involution



Boundary Logic Is Unorthodox

Boundary logic is not isomorphic to Boolean logic

- one-to-many map
- absence of relational concepts
- absence of arity and countability
- first class void-equivalence
- one basis constant
- two types of composability
- operational transparency (no function/argument distinction)

Boundary logic is not group theoretic

Identity for `SHARING`

```
a ◊ i  =  i ◊ a  =  a          identity
a   i  =  i   a  =  a          ◊ = SHARING
a      =      a  =  a          i = <void>
```

Inverse for `SHARING`

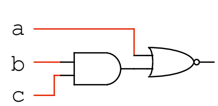
```
a ◊ a⁻¹  =  a⁻¹ ◊ a  =  i⁻¹    inverse
a  (a)   =  (a)   a  =  i⁻¹    ◊ = SHARING
   ( )   =  ( )      =  ( )    i⁻¹ = (i) = ( )
```

That is, the identity element, `i`, defined by the identity equations is the inverse of the identity element, i^{-1}, defined by the inverse equations.

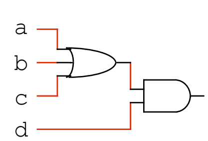
Table of Non-Correspondence

	BOOLEAN	BOUNDARY	DIFFERENCE
symbols	tokens	icons	linear vs spatial
constants	{0, 1}	{ () }	two vs one
unary operator	NOT	BOUNDING	delimited collection
binary operator	OR, AND	SHARING	not a function, not binary
arity	specific	any	no concept of argument
mapping	functional	structural	one-to-many
ordering	implicative	bounding	spatially explicit
computation	rearrange	delete	void-equivalence
semigroup	associative	no concept	boundary structure only
monoid	identity, i	<void>	existence
group	inverse	i^{-1}	new structure needed
Abelian group	commutative	no concept	no spatial metric

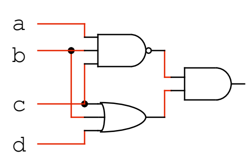
Circuit Structures in Boundary Logic



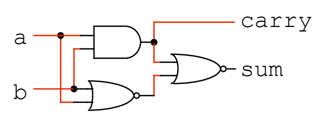
(a ((b)(c)))



((d)(a b c))



((b c d) ((a)(b)(c)))



sum = (carry (a b))
carry = ((a)(b))

Fully Nested Containment Graph

The containment graph representation is in Implicate Normal Form when there is no internal fanout (no structure sharing).

Implicate Normal Form	*Conjunctive Normal Form (PoS)*
deepest nesting	shallowest nesting (2 levels)
fewest literal references	most literal references
no internal pins	most internal pins
shortest wires	longest wires
unique up to form distribution	unique up to variable labeling
finesses intractability	grows exponentially large

4-bit Magnitude Comparator

```
((eq 1) (gt 2) (lt 3))
 ((1  ((j)(a (b))(b (a))(c (d))(d (c))(e (f))(f (e))(g (h))(h (g)))          )
  (2  ((i ((j)((g (h))((h (g))((e (f))((f (e))((c (d))(a (b)(d (c))))))))))  )
  (3  ((k ((j)((h (g))((g (h))((f (e))((e (f))((d (c))(b (a)(c (d))))))))))  )))
```


Abstraction and Management of Complexity

Design abstractions can be constructed bottom-up by using parens pattern templates.

Abstraction types

Functional modules, library cells
Structural modules, library macros
Dataflow modules, serial/parallel decomposition
Input symmetries
Parametric generation
Bit-width vector abstraction
Specialized technology maps (LUTs, FPGA cells)

Boundary logic transformations apply equal well to
- simple inputs (signals)
- compound boundary forms (subnets)
- modules and vectors (black-box abstractions)

Prototype Software Implementation

Software capabilities

- fully functional boundary logic reduction engines (logic, area, delay)
- functional tools for high-quality interactive netlist restructuring
- HDL language netlist parsers
- TSMC logic library mapping for delay and area models
- design exploration tools, including area/delay trade-offs

Software limitations

- prototype software implementation is proof-of-concept
- current LISP implementation is somewhat brittle
- delay modeling is based on weak physical models
- not an entire synthesis package
- not yet optimized for performance efficiency
- some functionality is designed but not yet implemented
- no user interface as yet

Conceptual limitations

- no personal HDL or layout design experience
- work has not been published, no peer review

Computational Pragmatism

Constructing a novel mathematical system is easy.
What differentiates good ones from bad ones is their utility.

Logic problems to calibrate computational utility:

Almost all problems found in logic textbooks are computationally trivial.

- simple tautology -- syllogistic dilemma

$$((a \rightarrow b) \wedge (c \rightarrow d) \wedge (a \vee c)) \rightarrow (b \vee d)$$

- simple minimization -- absorption

$$a \wedge (a \vee b)$$

- simple challenging tautology -- distribution of if-then-else (ite)

```
ite[ite[a,b,c], d, e] = ite[a, ite[b,d,e], ite[c,d,e]]
```

- simple challenging minimization -- factor forms with a dozen variables

  Reduce from 12 to 8 variable occurrences:

$$(\neg a \wedge (\neg(g \vee (b \wedge c)) \vee \neg(f \vee (d \wedge e \wedge g)))) \vee \neg((b \wedge c) \vee (d \wedge e))$$

- commercial tautology

  c5315 -- 178 inputs, 123 outputs, 1300 logic gates
  80386core -- 36 inputs, 70 outputs, 20000 logic gates

- commercial minimization

  minimal area and delay for above circuits

- huge randomly constructed SAT problems

  10,000s of variables, millions of clauses