THE ADVANTAGES OF BOUNDARY LOGIC -- A COMMON SENSE APPROACH

William Bricken
December 2002, updated July 2004

Containers

Boundary logic uses nested containers to express logic and computation. Think of any physical container such as a glass, a box, or a room. A common sense example of nested containers is a piece of candy inside a wrapper inside a box inside a shopping bag.

A container has an inside and an outside. In text, any delimiter is a container. Eg:

```
outside ( inside )
                  \
                   delimiter
```

Boundary logic shows that Boolean logic is simply configurations of nested containers. The relationship of containment expresses everything about logical forms.

```
( ( ) )              two containers, one inside the other

( ) ( )              two containers in the same space
```

> *Common sense example:* How many ways can a box and a glass be arranged? They can be outside each other, or one can be inside the other. The two ways they are arranged can be used as two different logic values.

Boundary logic is much simpler than Boolean logic, since *one* container is the same as *two* Boolean values. Boolean logic uses two different tokens for values, while boundary logic uses one container and two different spaces.

```
   0, 1                 two different Boolean values

   (   )                one container
    /   \                      and
inside   outside          two different spaces
```

Boundary logic is *unary*, Boolean logic is *binary*. Unary logic has only one type of value and one type of structure (the container), and is therefore *half as complicated* as binary logic.

Here is how boundary logic manages to be simpler than Boolean logic:

Either space distinguished by a container can be empty. Empty spaces require no processing, because there is nothing there to process.

(A)	nothing on the outside
( ) A	nothing on the inside

Boolean logic symbols can never be "not there", so they can never be simplified by not existing. Boundary logic uses "nothing" to simplify computation.

Algebraic Transformation

Boundary logic is *algebraic*, the same as the elementary algebra of numbers. That is, boundary transformations work by following specific substitution rules. The pattern of containers on one side of a rule is identified, and the other side of the rule is substituted for that pattern. Pattern-matching and substitution are very simple to implement, and use well-understood low effort algorithms.

((A)) = A	the **Involution** rule
(B ((C D))) ==> (B C D)	an application example

In boundary logic, letters are variables, just as in the algebra of numbers. A letter stands in place of any configuration of containers and other letters.

> *Common sense example:* Two nickels equal one dime. If you have two nickels in your pocket, you can exchange them for a dime without changing your wealth, but you have less stuff in your pocket.

Logic Reduction

Reduction using deletion or erasure

Because spaces support "nothing", boundary transformations work by *deleting structures*. Boolean transformations work by accumulating and rearranging structures. Deletion has excellent computational properties: the problem gets smaller for each rule application, thus processing gets faster while problem size decreases.

(A ( )) = "nothing"   the **Occlusion** rule

(B (C ( ))) ==> (B   )   an application example

> *Common sense example:* You are in a room. You walk out the door and then back in the door. You can delete, or not perform, the two passages through the door because you end up in the same place.

No ordering, no grouping, no counting

Containers can exist anywhere within a given space. There is no implicit ordering so you don't need to keep track of what is first and what is second. The sequence of forms in a logic problem is represented by the depth of nesting of containers, while all structures within a given space can be processed in parallel.

Logical operators have a specific number of arguments, usually two. Containers have any number of contents. Boundary logic does not require you to keep track of how many arguments there are because it does not matter how many structures are inside a container.

Similarly, logical operators assemble their arguments into groups. A container also groups all the forms inside it. However, since the container accommodates any number of forms, there is no grouping of those structures other than that imposed by the container they are in.

> *Common sense example:* There are many people in a movie theater. To the movie, it doesn't matter how many people, or where they are sitting, or what groups of other people they came in with (boundary). Boolean techniques keep track of who is where and when they came in.

Reduction of paths rather than operators

Containers are transparent from the outside: when structures are outside a container, they can be deleted from the inside. This applies regardless of the depth of nesting of containers. Conventional logic reduces every operator separately. Boundary logic can often ignore intervening containers to optimize an entire path of logic in one step.

A (A B) = A (B)   the **Pervasion** rule

C (D (C E))) ==> C (D (E))   an application example

> *Common sense example:* You have an open box of candy in a shopping bag. You can reach in to get a piece of candy without being obstructed by the shopping bag (boundary). Boolean techniques treat the shopping bag and the box as always closed.

Only one type of structure, the container

Logical forms are composed of operators and arguments. The separation of operation and connection (logic and wiring) through arguments is an artifact of conventional logic. In a boundary approach, logic and connectivity are *the same thing*. This effectively eliminates the step-wise approach of Boolean logic.

> *Common sense example*: Your car is parked. You get in and drive it away. From the outside, the car is either still or moving (Boolean). But when sitting on the inside, the dashboard is always still (boundary).

Explicit parallelism

Structures sharing the interior of a container are *independent* and thus can be processed in parallel. Boolean techniques are inherently sequential.

> *Common sense example:* You are looking for the ace of spades in a deck of cards. You turn over one card at a time until you find it (Boolean). Or you turn over the entire deck all at once and spread it out on the table to find the ace (boundary).

Formality throughout

Boundary logic is based on transformation rules; a boundary logic transformation system that follows the rules is always correct, providing built-in verification.

> *Common sense example:* You are walking down a hallway. The walls make you enter the rooms only through doors. You can't make the mistake of walking through a wall (formal). Or you can walk into the wall and risk a broken nose (not formal).

One-to-many mapping

One way to describe the power of boundary logic is to say that one boundary form is equivalent to many different Boolean forms, and one boundary transformation is equivalent to many Boolean transformations.

(A) boundary form for NOT A,
and also for A IMPLIES FALSE
and also for NOT A OR FALSE

Common sense example: You withdraw ten dollars from the bank (boundary). The teller can give it to you in many different combinations of bills (Boolean).

Nothing more

The three rules identified above (Involution, Occlusion, Pervasion) define *all* basic boundary transformation rules, and thus comprise all possible legal transformations. Each rule identifies a pattern on the left-hand-side and a simpler substitution pattern on the right-hand-side that deletes something. These three deletion rules are sufficient for logic reduction.

History

Boolean logic has been embedded in language since antiquity. When computers were invented, it was natural to adopt the linear, sequential characteristics of Boolean logic found in language. However, semiconductor circuitry works in parallel, more like a group of people than a string of words.

Boundary logic was discovered over 100 years ago, by the founding fathers of formal mathematics (Peirce, Frege). The world elected to follow *binary symbolic logic* until now, essentially ignoring the efficiencies of boundary logic. This choice has been so extreme that boundary logic is completely unknown to the wider community of scholars.

Summary

Boundary logic translates Boolean logic into a form that is consistent with computation (parallel) rather than talking (serial). The result is a logic reduction tool set that is:

-- much simpler to use
-- much simpler to implement
-- provides unique new tools for logic reduction and optimization
-- suggests new computer architectures that may be far more efficient
-- has the potential to improve all computational techniques.