PULLING A RABBIT OUT OF THE HAT

William Bricken
June 2001

Boundary logic is *unary* rather than binary; it uses one enclosing iconic form {O}, rather than two symbolic forms {0,1}.

Inside One

The way this is accomplished is absolutely simple, although the idea is both innovative and unfamiliar. Envision the number One:

```
                  1
```

Our viewpoint stands outside of the printed page, and outside of the symbol 1. Although it has an outside, the symbol 1 does not have an inside. Iconic forms do have an inside, they are two dimensional. We have simply given One an inside by representing it as an enclosure, or container:

```
                  1 = ( )
```

The inside of One is completely empty, no new properties have been added other than *insideness*.

The inside of One is *not* the outside. In binary computation, the symbol Zero 0 represents that which is not One. In unary computation, the empty inside of One acts as Zero, it is not-One. However, there is nothing recorded inside One, the idea of zero consists of leaving the inside of One empty. Absence, of course, is not a representation and does not occur as a data-structure. Thus Zero has no representation, it is accessed indirectly by looking inside One. Thus we have One and Nothing-Inside-of-One instead of 1 and 0. The token 0 is not needed during computation, it occurs only at output, in order to transcribe unary logic back into binary logic

```
            1 is            .  (  .  )
                           /        \
            has an outside           has an inside
```

Outside, we see ( ) and we see 1. Both exist as objects.

Inside, we see nothing and we infer it is 0. Zero only exists as a symbol.

Even though the iconic One is empty inside, that void (like the imaginary rabbit Harvey) can be used to make things easier.

Filling One

What happens when there is something inside an iconic One? What *can* be inside One?

Thus far, the iconic One is the only representation we have. (We do not yet have a rabbit to put inside.) At this early stage, the inside of One can either be empty, or it can have another One inside it:

```
1  =  (  )             nothing inside
2  =  (())             One inside
```

As a number, the iconic boundary can be seen to double its contents. Iconic forms have a place notation also. Rather than keeping track of "spaces to the left" as we do for binary string power notation, we simply count depth of nesting to find out the power. The rule of iconic depth notation is that deeper is larger.

In the case of logic, the ideas of Two and Double do not exist. When we fill an iconic logic enclosure with a replicate of itself, we change from True to False:

```
True  = (  )
False = (()) = <void>
```

Unary to Binary

Current computers are binary. How do we accomplish unary computation on binary machines? The map is again simple:

```
1 = (                  < (   seen from outside
0 = )                  < )   seen from inside
```

Each "side" of the linearized parens enclosure is encoded as in binary. The cost is that the spatial, parallel nature of enclosures is forced into a linear string representation. An example:

```
Parens form:     (()(()))
Binary form:     11011000
```

Each parens form expresses a unique logical configuration. The accompanying binary string is also unique. As well, the string has a parity, since 1s and 0s are balanced in well-formed parens configurations.