LOSP OVERVIEW

William Bricken
December 1999

Losp is a suite of three hierarchical software tools. At the core, the **Boundary Logic Engine** provides Boolean minimization of logic function graphs. The **Circuit Graph Engine** manipulates circuitry-specific transformations on structure (such as technology library mapping). The outer layer of functionality, the **Application Interface**, admits and generates EDIF specifications, provides batch-mode processing, test-vector verification, statistical analysis, and interfaces to CAD and other interactive systems.

The main innovation in Losp is based on a representational credo: less is better. Boundary Logic uses *void-based techniques* extensively, to reduce and to simplify both representation and transformation of circuits expressed as graphs.

Void-based techniques assign meaning to the absence of a representation, to non-representation, essentially eliminating half of the representational form. An example: it takes one label to differentiate two bags. Similarly, the binary system can be represented with one explicit and one implicit token: Let a mark represent 1 and the absence of a mark represent 0.

To make this work, the mark must have the ability to *contain*. The essential idea is that an empty container can indirectly refer to an absence of contents, to a void. Containing forms permit void reference without introducing ambiguity. Containers are typographically represented by delimiting pairs of tokens, such as ( ). Let the mark ( ) = 1 and the contents of the empty mark <void> = 0. The mark then negates (as well as contains) the void.

The same void-based simplification can be applied to Boolean functions. Let the absence of a connecting mark represent disjunction, OR. From this infrastructure, a mark which contains two tokens, such as (a b), represents a binary NOR gate with inputs a and b. Taking the disjointness of space literally, we can let any spatial arrangement of any number of forms carry the associativity, commutativity, and n-arity of OR.

The use of the void reduces the minimal basis for propositional logic from two textual tokens {if, false} to one containing token {mark}, without loss of expressability.

As well as providing a reduced notation, void-based forms have nice transformational properties:

• Evaluation is through deletion, or erasure, of structure. Generation of new facts (ala resolution) is replaced by erasure of irrelevant forms.

• Forms which are in the equivalence class of the void can be freely erased and freely inserted wherever desirable.

• Due to the pervasiveness of void within any representation, transformations treat depth of nesting as transparent.

Here is the body of a small combinational circuit (ISCAS'89 c17), represented as a logic function graph:

```
( (8   (9 (d))           )               d and not 9
  (9   ((b) (c))         )               b and c
  (4   ((8 (9 (a))))     )               8 or (a and not 9)
  (5   ((8 ((c) (e))))   ) )             8 or (c and e)
```

On each line, the left hand number identifies a logic node, the right hand form is that logic. The right-hand forms encode logic in nested and adjoined parens. (The conventional logic is written on the right.) These list data structures are extremely easy to manipulate during pattern-matching. Three axioms in a "boundary equational logic" define the computational rules:

```
Dominion:      ( ) A = ( )

Involution:    ((A)) = A

Pervasion:     A (A B) = A (B)          A (..(A B)) = A (..(B))
```

Capital letters stand for arbitrary logical or circuit forms. These rules have nice substitution and convergence properties, a consequence of using void-substitution, or erasure, as a primary transformation technique. An example of a Boundary Logic proof follows:

```
Prove:  A and (A or B) = A          ((A)(    A B)) = A      transcribe
                                    ((A)((A) A B))          pervasion (A)
                                    ((A)(( ) A B))          pervasion A
                                    ((A)(( )    ))          dominion
                                    ((A)         )          involution
                                      A                     involution, QED
```

The Boundary Logic Engine applies these three transformation rules. Pervasion is more complex than the others, since replicated forms can be erased from any depth of nesting. Pervasion also provides an innovative computational technique: *virtual insertion*. All outer forms (higher in the graph/circuit) can be freely inserted into all lower branches. *Virtual* means that the form is arbitrarily present or not, depending on its usefulness for reduction. Once the virtual form and its context have been reduced, any remaining parts of the virtual form can be freely erased, since (by virtue of their void-equivalence) they can have no effect on the value of the logic graph.

Due to the homogeneous representation, patterns in the circuit can be easily identified. The Circuit Graph Engine identifies common gate patterns (XOR, MUX), common structures which can be shared by increasing fanout (structure sharing), and common patterns found in technology libraries (technology mapping).

Losp has been applied to the ISCAS’89 benchmarks, for both combinational and sequential circuitry. On about half of these circuits, the Losp minimization algorithm produces world-class results.