HOW DOES LOSP WORK?

William Bricken
March 2001

Losp consists of two systems, the BM reduction (parens) algorithms, and the circuit management (pun) algorithms.

TOP-LEVEL

1. Transcribe the EDIF specification literally into the equivalent pun format.

2. If possible, remove all intermediate circuit nodes, leaving a pure parens form of the circuit.

    a. Apply BM reduction algorithms to the parens circuit, or
    b. Apply BM reduction to each node of the pun circuit.

3. Apply abstraction and structure-sharing transforms to the pun form.

4. Convert the final result back into EDIF.

BM REDUCTION

The parens engine reduces only parens forms. It applies the axioms of BM until no further reduction.

Axioms

A ( ) = ( )	DOMINION
((A)) = A	INVOLUTION
A (A B) = A (B)	PERVASION

Since the BM axioms can be implemented by pattern-matching, it is possible simply to enter the axioms into a declarative reduction engine (such as Prolog or any expert system). This approach is extremely inefficient.

The current Losp implementation mixes axiom applications opportunistically, using a pure functional rather than declarative approach. The steps, all incorporated into one two page algorithm, follow:

1. Put parens into lexicographic canonical form.

2. Apply DOMINION and INVOLUTION to remove all grounds and redundant parens.

3. Apply PERVASION in successively stronger forms:

   a. EXTRACT literals from all depths
   b. EXTRACT and SUBSUME bounded forms from all depths.
   c. VIRTUAL INSERTION of pairs of bounded forms.

4. Distribute and partially distribute the remaining form to minimize variable references.

((a b)(a c)) = a ((b)(c))   DISTRIBUTION

The VIRTUAL INSERTION algorithm is unique, in that it uses a query form which descends into the deepest space of the target form. As the query form passes a given space, all forms in that space are EXTRACTED. If the query form itself reduces to ( ), then that mark acts through DOMINION on the current context to eliminate that context entirely. During this process, the target form reduces solely through query form erasure, so that no additional assertions and textual rearrangements are required.

PUN REDUCTION

As well as BM reduction, the pun form of the circuit itself can be simplified. The pun form is simply a collection of threaded parens forms; the parens form broken into cells/nodes due to either problem size or to repetition of subforms. Steps:

1. Literal EDIF parse.

2. Remove constants and inverters.

3. Expand any node with a single reference in the body of the circuit.

4. Rename and rearrange nodes.

The above four steps are for preliminary structuring of the pun form, and do not require smart ordering of nodes or operations. Ordering is critical for the next steps:

5. Expand references which reduce in their new context.

6. Abstract XOR/IFF and ITE forms.

7. Coalesce shared structure.

All parens and pun reduction algorithms use a nested list as the logic data structure. With parens as the only (spatial) operator, pattern-matching is quite easy and efficient, and parsing is unnecessary.

Example

A simplified example illustrates these steps:

```
(main
  ((a unk) (b unk) (c unk))         inputs and bindings
  ((oa 1) (ob 5))                   outputs and bindings
  ( (1 (2 3 (4) 5))                       body
    (2 (a b))
    (3 (c))
    (4 ((a)(b)))
    (5 (4 (a b)))  ))
```

STEP 2: remove node 3, a inverter

```
  ( (1 (2 (c) (4) 5))               body
    (2 (a b))
    (4 ((a)(b)))
    (5 (4 (a b)))  ))
```

STEP 3: remove node 2, a single reference

```
  ( (1 ((a b) (c) (4) 5))
    (4 ((a)(b)))
    (5 (4 (a b)))  ))
```

STEP 5: expand references which reduce

```
  ( (1 ((a b) (c) (((a)(b))) (4 (a b))))        ==>  (1  ((c)(a)(b)))
    (4 ((a)(b)))
    (5 (4 (a b)))  ))
```

STEP 6: abstract EQ/ITE

```
  ( (1 ((c)(a)(b)))
    (4 ((a)(b)))
    (5 (((a)(b)) (a b)))  ))        ==>  (=5=  ((((a)(b)) (a b)))
```

STEP 7: coalesce

```
  ( (1 ((c)(4)))
    (4 ((a)(b)))
    (=5= ((((a)(b)) (a b))))  ))
```