EXAMPLES OF PUN HIERARCHICAL DATA STRUCTURE

William Bricken
June 1997

```
=================
    3MAJORITY
=================
```

Generating the function with tables:

```
      clause = conjunction of signed variables
      function = disjunction of clauses
```

a	b	c	fn	clauses
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	(a (b)(c))
1	0	0	0	
1	0	1	1	((a) b (c))
1	1	0	1	((a)(b) c)
1	1	1	1	((a)(b)(c))

```
      fn = (( (a (b)(c)) ((a) b (c)) ((a)(b) c) ((a)(b)(c)) ))
```

Representing the clausal form of this function in pun:

```
    ((3majority)
     ((main)
      ((a unk) (b unk) (c unk))
      ((oa 0))
      ((0  (8) )
       (1  (a) )
       (2  (b) )
       (3  (c) )
       (4  (a 2 3) )
       (5  (1 b 3) )
       (6  (1 2 c) )
       (7  (1 2 3) )
       (8  (4 5 6 7) )   )))
```

A functionally equivalent pun form, reduced by BM:

```
    ((3majority)
     ((main)
      ((a unk) (b unk) (c unk))
      ((oa 0))
      ((0  (4 5 6) )
       (4  (b c) )
       (5  (a c) )
       (6  (a b) )   )))
```

Here the cells are completely expanded:

```
    ((3majority)
     ((main)
      ((a unk) (b unk) (c unk))
      ((oa 12))
      ((12  ((a b) (a c) (b c)) )   )))
```

Partial evaluation (dynamic reduction) of the form generates
a new (reduced) function:

```
    ((3majority)
     ((main)
      ((a unk) (b 0) (c unk))
      ((oa 12))
      ((12  ((a  ) (a c) (  c)) )  )))

  ==>

    ((3majority-part)
     ((main)
      ((a unk) (c unk))
      ((oa 12))
      ((12  ((a) (c)) )  )))
```

```
============
    DAIO
============
```

Result of EDIF2PUN0:

```
    ((daio)
     ((main)
      ((clok unk) (a unk))
      ((oa 9) (ob 0))
      (( 0  (((clok) 1)) )
       ( 1  ((3 2)) )
       ( 2  ((6) (0)) )
       ( 3  ((5) (4)) )
       ( 4  (15) )
       ( 5  (6) )
       ( 6  (((clok) 7)) )
       ( 7  ((9 8)) )
       ( 8  (6) )
       ( 9  ((15) (10)) )
       (10  ((14 11)) )
       (11  ((13) (12)) )
       (12  (21) )
       (13  (a) )
       (14  ((a) (21)) )
       (15  ((19 16)) )
       (16  ((18) (17)) )
       (17  (20) )
       (18  (21) )
       (19  ((21) (20)) )
       (20  (((clok) 21)) )
       (21  (((clok) a)) )   )))
```

Cells reduced by eliminating single variables and single references:

```
    ((daio)
     ((main)
      ((clok unk) (a unk))
      ((oa 9) (ob 0))
      (( 0  ((1 (clok))) )
       ( 1  (((6 15) ((0) (6)))) )
       ( 6  ((7 (clok))) )
       ( 7  ((9 (6))) )
       ( 9  ((15) ((21 a) ((21) (a)))) )
       (15  (((20 21) ((20) (21)))) )
       (20  ((21 (clok))) )
       (21  ((a (clok))) )   )))
```

New cells introduced by pattern labeling:

```
    ((daio)
     ((main)
      ((clok unk) (a unk))
      ((oa 9) (ob 0))
      (( 0     ((1 (clok))) )
       ( 1     (((6 20=21) ((0) (6)))) )
       ( 6     ((7 (clok))) )
       ( 7     ((9 (6))) )
       ( 9     ((20=21) (a=21)) )
        20     ((21 (clok))) )
       (21     ((a (clok))) )
       (20=21  (((20 21) ((20) (21)))) )
       (a=21   (((a 21) ((a) (21)))) )   )))
```

New library component (named eq) introduced by pattern abstraction:

```
    ((daio)
     ((main)
      ((clok unk) (a unk))
      ((oa 9) (ob 0))
      (( 0      ((1 (clok))) )
       ( 1      (((6 20=21-0) ((0) (6)))) )
       ( 6      ((7 (clok))) )
       ( 7      ((9 (6))) )
       ( 9      ((20=21-0) (a=21-0)) )
       (20      ((21 (clok))) )
       (21      ((a (clok))) )
       (20=21-  (eq ((a 20)(b 21)) ((oa 20=21-0))) )
       (a=21-   (eq ((a  a)(b 21)) ((oa  a=21-0))) )))
     ((eq)
      ((a unk) (b unk))
      ((oa 0))
      ((0  (((a b) ((a)(b)))) )   )))
```

Between-register combinational cells expanded, forming alternating registers and logic blocks. The (clok) token stops further expansion of cells.

```
    ((daio)
     ((main)
      ((clok unk) (a unk))
      ((oa 9) (ob 0))
      (( 0  ((1 (clok))) )
       ( 1  (((6 ((20 21) ((20) (21)))) ((0) (6)))) )
       ( 6  ((7 (clok))) )
       ( 7  (((6) (((20 21) ((20) (21))) ((a 21) ((a) (21)))))) )
       (20  ((21 (clok))) )
       (21  ((a (clok))) )   )))
```

Time-indexed form used for retiming. The (clok) label advances the *local-time* by one tick. All cloks tick in unison. The pun notation is close to that of recursive function theory. In applications, the index "i" is always bound to a specific integer.

```
    ((daio-timed)
     ((main)
      ((clok.i unk) (a.i unk))
      ((oa 9) (ob 0))
      (( 0.i+1  ((1.i (clok.i))) )
       ( 1.i    (((6.i ((20.i 21.i) ((20.i) (21.i)))) ((0.i) (6.i)))) )
       ( 6.i+1  ((7.i (clok.i))) )
       ( 7.i    (((6.i) (((20.i 21.i) ((20.i) (21.i)))
                         ((a.i 21.i) ((a.i) (21.i)))))) )
       (20.i+1  ((21.i (clok.i))) )
       (21.i+1  ((a.i (clok.i))) )   )))
```

Between-register blocks expressed as library components. The "n-" cell label identifies library calls. Top and bot bindings in the library call thread the library subgraph into the main graph.

```
    ((daio)
     ((main)
      ((clok unk) (a unk))
      ((oa 9) (ob 0))
      (( 0  ((1-0 (clok))) )
       (1-  (t1 ((a 6)(b 0)(c 20)(d 21)) ((oa 1-0))) )
       ( 6  ((2-0 (clok))) )
       (2-  (t2 ((a 6)(b a)(c 20)(d 21)) ((oa 2-0))) )
       (20  ((21 (clok))) )
       (21  ((a (clok))) )))
     ((t1)
      ((a unk)(b unk)(c unk)(d unk))
      ((oa 0))
      ((0  (((a ((c d) ((c) (d)))) ((a) (b)))) )))
     ((t2)
      ((a unk)(b unk)(c unk)(d unk))
      ((oa 0))
      ((0  (((a) (((c d) ((c) (d))) ((b d) ((b) (d)))))) )   )))
```

Recursive embedding of library abstractions:

```
    ((daio)
     ((main)
      ((clok unk) (a unk))
      ((oa 9) (ob 0))
       (0   ((1-0 (clok))) )
       (1-  (t1 ((a 6)(b 0)(c 20)(d 21)) ((oa 1-0))) )
       (6   ((2-0 (clok))) )
       (2-  (t2 ((a 6)(b a)(c 20)(d 21)) ((oa 2-0))) )
       (20  ((21 (clok))) )
       (21  ((a (clok))) )))
     ((t1)
      ((a unk)(b unk)(c unk)(d unk))
      ((oa 0))
      ((0   (((a (1-0)) ((b) (a)))))
       (1-  (eq ((a a)(b b)) ((oa 1-0))) )))
     ((t2)
      ((a unk)(b unk)(c unk)(d unk))
      ((oa 0))
      ((0   (((a) (1-0) (2-0))) )
       (1-  (eq ((a c)(b d)) ((oa 1-0))) )
       (2-  (eq ((a b)(b d)) ((oa 2-0))) )))
     ((eq)
      ((a unk) (b unk))
      ((oa 0))
      ((0  (((a b) ((a)(b)))) )   )))
```

Circuit partitioned into two variable cells, for 2LUT mapping:

```
    ((daio-3var)
     ((main)
      ((clok unk) (a unk))
      ((oa 9) (ob 0))
      (( 0    ((1 (clok))) )
       ( 1    ((2 3)) )
       ( 2    (6 20=21) )
       ( 3    ((0) (6)) )
       ( 6    ((7 (clok))) )
       ( 7    ((8 (6))) )
       ( 8    ((20=21) (a=21)) )
       (20    ((21 (clok))) )
       (21    ((a (clok))) )
       (20=21 (((20 21) ((20) (21)))) )
       (a=21  (((a 21) ((a) (21)))) )   )))
```