CYCLIC STRING NOTATION

William Bricken
January, 1995

PART I

Sequential circuits are directed cyclic graphs (DCGs) which maintain both a logical semantics (the functionality of the combinatorial circuit) and a temporal semantics (the effect of feedback of logical values on the functionality of the circuit). This memo presents an approach to the representation and transformation of DCGs with these desirable properties:

- logical and temporal invariants expressed as axioms and theorems
- sequential circuits expressed as structured strings
- DCG optimization implemented by string rewriting

String-based DCG transformation integrates tightly with existing Losp tools.

DCGs interpreted for logic can be represented as circuit diagrams and as distinction networks. To represent the process of circuits, the representation must be animated. There is a clear division between the representational needs of the interface (design, interaction, understanding) and the needs of the computational engine (evaluation, transformation, optimization). The focus here is on structural transformation of sequential circuits at the engine level.

Applications of DCG axiomatic transformation include:

- placement and rearrangement of registers and combinatorial blocks
- optimization of structure and timing
- hierarchical modularization.

The notes which follow explore the space of DCG structures, some elementary reduction axioms, and some counting principles. This work has not yet been applied to practical circuits, although the interpretation for circuits is clear.

DCG Notations

In the following, parens represent distinctions, with standard boundary logic interpretations for combinatorial circuits. Each parens (distinction node) is labeled.

A circuit is expressed in PUN (parens unexpanded) form, as labeled subgraphs with reentry. [See the memo "Combinatorial Strategy" (8/95) for PUN representation and transformation.] For example, the elementary imaginary circuit is expressed in PUN as:

```
out = (in)
in = out
```

PUN can be expanded to cyclic string notation by substituting the algebraic labels:

```
out = (out)
```

The temporal semantics of this structure can be expressed by indexing:

```
out0 = <init>
out1 = (out0)
...
outN = (outN-1)
```

A critical issue is the mixture between PUN and expanded forms. Cyclic string notation was developed to permit flexibility in shifting between levels of expansion.

Example

For a more complex example, consider the naive counter in Shoup's Transition Logic memo (notation is slightly modified but the techniques are strongly associated):

```
0 = (in)
1 = (in 8)
2 = (in 7)
3 = (1 4)
4 = (2 3)
5 = (0 3)
6 = (0 4)
7 = (5 8)
8 = (6 7)
out = 7
```

This PUN form has plenty of reentry, but expansion/substitution steps both simplify the representation and expose the symmetries. Note that substitution here has the semantics of tracing a signal through the entrant circuit. Thus, expansions/substitutions which do not force reentries in the same equality are simply changing dnode *names* from explicit to implicit form.

Eliminate {0, 1, 2, 5, 6}:

```
3 = ((in 8) 4)
4 = ((in 7) 3)
7 = (((in) 3) 8)
8 = (((in) 4) 7)
out = 7
```

Eliminate {4, 8} [Flagg Resolution, must substitute for all occurrences]:

```
3 = ((in (((in) ((in 7) 3)) 7)) ((in 7) 3))
7 = (((in) 3) (((in) ((in 7) 3)) 7))
```

Further substitution now iterates the circuit temporally. Note that no information or structure or functionality is lost across these different expansions. Also note some choice of what basis nodes to use, for eg:

Eliminate {3, 8}

```
4 = ((in 7) ((in (((in) 4) 7)) 4))
7 = (((in) ((in (((in) 4) 7)) 4)) (((in) 4) 7))
```

The Cutting Edge

For simplification of the circuit, the important issue is now: What are the valid transformations, the axioms of reentry?

We have not yet studied this issue in depth, but both the Varela and the Shoup systems are candidates. As an example, the Varela calculus has the following axioms:

```
((X) Y) X    = X
((X Y)(X Z)) = X ((Y)(Z))
(X i) X      = X i
```

where X can be high, low, or reentrant, and i is the elementary oscillating/imaginary form. By permitting variables to be reentrant, string transformations on expanded circuits can be made without violation of invariants.

PART II

To explore cyclic string notation and DCGs in general, an exhaustive listing of simple DCGs was generated and forms were counted. Samples of this listing follow:

All DCGs:

innermost parens is named i
outermost parens is named j

<void>

()

(i)
i()

i(i)

(())

((i))
(i())
i(())
((j))
(j())
j(())

(i(i))
i((i))
i(i())
(j(j))
j((j))
j(j())

(j(i))
j((i))
j(i())
(i(j))
i((j))
i(j())
((ij))
(ij())
ij(())

(((())))

etc.

The above listing incorporates some valid reentrant circuit reduction rules, in particular,

i i = i duplicate feeds of the same signal do not change functionality

i j = j i ordering of feeds does not change functionality

The growth of forms is similar to that of the binomial expansion. These forms can be counted in respect to their length, counting parens pairs as length 1.

Table entry is number of possible cyclic circuits

v Number of chars N \ Number of parens in expression:

	n	1	n-1	2	n-2
1	1				
2	1	2			
3	1	3	6		
4	1	4	12	24	
5	1	5	20	60	90

CASE of N chars, parens increase from 0 to N:

```
1                       the <void>

/N\ * 1^(N-1)           one paren
\1/

/N\ * 2^(N-2)
\2/

/N\ * 3^(N-3)
\3/

...

/N\ * i^(N-i)           i parens
\i/
                        when N is even next term is identical
/ N \ * (N-i)^i
\N-i/

 ...

/ N \ * (N-2)^2
\N-2/
```

```
/ N \ * (N-1)^1
\N-1/

1                       N parens
```

Generally, the number of DCGs is:

```
            SUM from i=1 to N of    /N\ * i^(N-i)
                                    \i/
```

where N is string length (counts reentries)
and i is number of dnodes.

This count decreases by the number of functionally invariant forms (as defined by the axioms of the system). For example:

A: same parens cannot enter the same space more than once
B: two or more different parens entering the same space do not have ordering
C: entry at top level of self is spatial duplication (i.e. cardinality)

Summary:

This exploration represents, counts and classifies DCGs in preparation for axiomatic transformation of sequential circuits.