Analysis of Hypercube Tree Topology

This paper has been inspired by a number of sources, a culmination of varying subjects. However, the more recent and most notable sources for subjects used in this paper are listed below.

Matt Parker - Stand-Up Maths

http://standupmaths.com/

Hypercube and Turing Machine Videos

Moritz Firsching - Unfolding of the Hypercube

https://unfolding.apperceptual.com/

Pascal Michel - Busy Beaver Game (Explanation)

https://webusers.imj-prg.fr/~pascal.michel/

The inspiration for this paper came from thinking about how to analyse Turing state machine instruction sets, one comment in one of my Turing state machine scripts was this:

"*# tree build(?)*

 # analyse paths and build a tree diagram (?)"

Fast forward to watching a video about unpacking hypercubes by Matt Parker analysing Moritz Firsching results.

The results are interesting due to the tree diagrams, I had not understood to this point how tree diagrams could be represented as instruction sets in state machines.

This paper walks through an example of mapping Tree Set 1 from Moritz Firsching paper into a Turing State Machine.

A nod to Pascal Michel at this point, as primarily that website was my source of reference for understanding and building a Turing machine.

Version

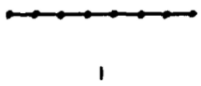
Version	Changes
v1	First draft
v2	Correction in combining the tracks from Figure 6.1 -> End of File. Diagrams updated. Assumptions on how to combine in text updated around associated diagrams.
v3	Pasted wrong diagram result matrix for subtraction

Tree Diagram Analysis

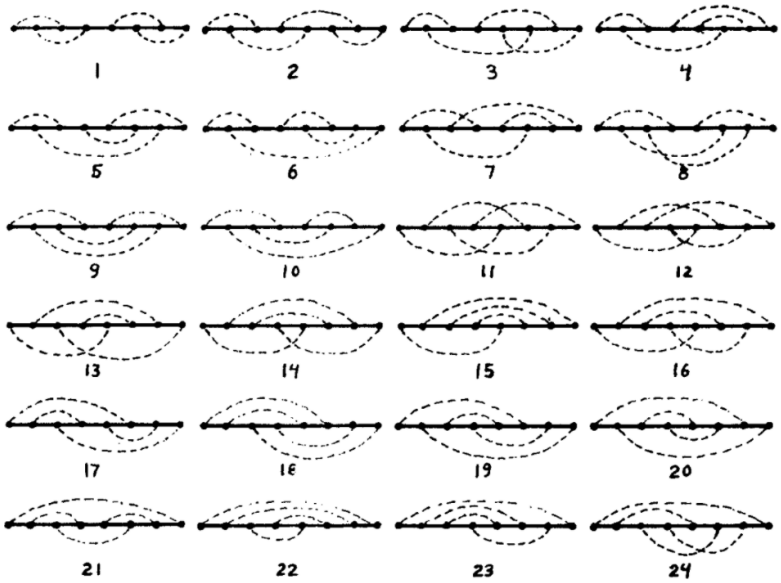
Taking Moritz Firsching tree diagrams this paper seeks to aim analysis at a particular “Tree Set”.

https://unfolding.apperceptual.com/

In figure 10 Tree Set 1 is described as having this topology:



In figure 11.1 trees within the set are described:



Tree Set 1 Rationalisation

Consider the Tree can be symmetrical along both X and Y.

Tree 1

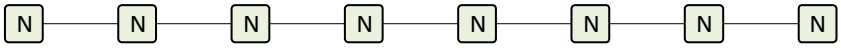


Figure 1.1

Tree 1 Potential 1 (6 Choices) (not 12 choices due to symmetry)

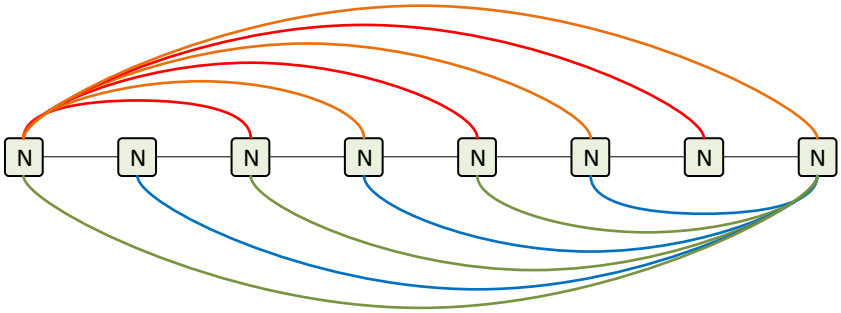


Figure 1.2

Tree 1 Potential 2 (4 Choices) (not 8 choices due to symmetry)

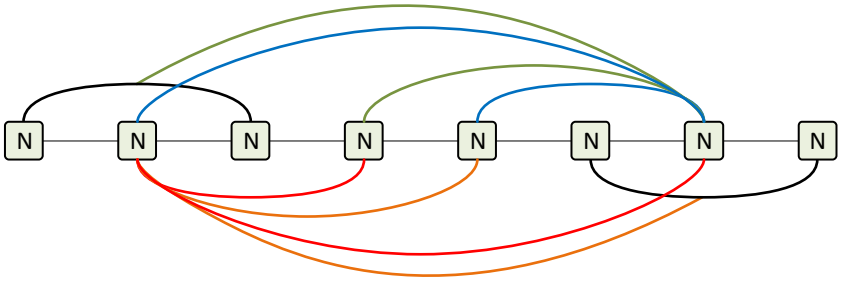


Figure 1.3

Resultant Tree 1 (6 x 4 = 24)

Consider that N can be any number. Any arrangement of nodes can have the numbers rearranged to satisfy this same model.

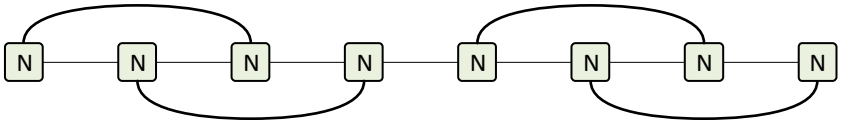


Figure 1.5

For Example: This satisfies Tree 1.2

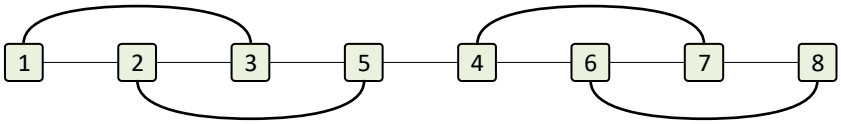


Figure 1.6

Tree Set 1 Vertices Inclusion in Topology

Tree 1 Vertices Potential 1 (2 Choices)

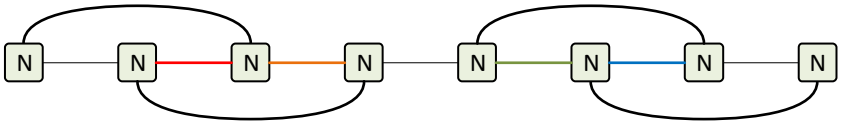


Figure 1.7

Tree 1 Vertices Potential 2 (1 Choice)

Resultant Tree 1 (24 x 2 x 1 = 48)

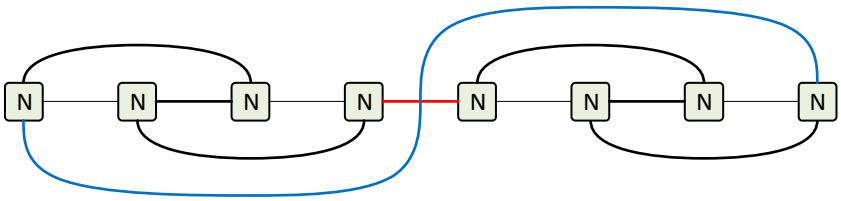


Figure 1.8

Mapping the Topology onto a Turing Machine

Consider a sigma(4,2) turing machine. I believe it is possible to do similar in a sigma(3,3) (which does produce some interesting numbers).

Turing Machine

We substitute the end of the topology diagram with the H (Halt) instruction. Otherwise we need to step up to use a sigma(3,3) where the 9th node would be an imaginary point outside of the tree that would need to be included in the potential mappings of the tree and calculated in accordingly.

This paper seeks to analyse the sigma(4,2) state machine.

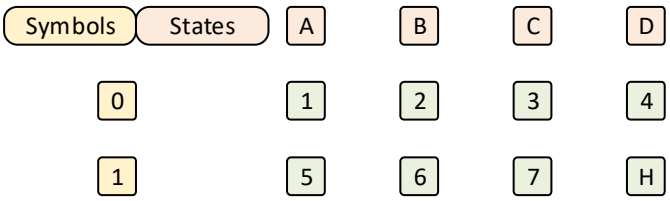


Figure 2.1

Turing Machine Mapping Potentials

Each green node represents the nodes on the topology tree diagram. Each diagram represents a rotation of the green nodes around the state machine. The shift moves all nodes around one place to the right each iteration.

Each yellow symbols represents the symbol of the state machine - in this case 0 and 1.

Each orange node represents the states of the state machine - in this case A, B, C, and D.

Black connections represent the face connections and blue the vertices connections.

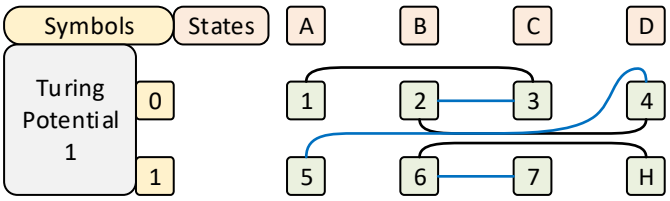


Figure 2.2

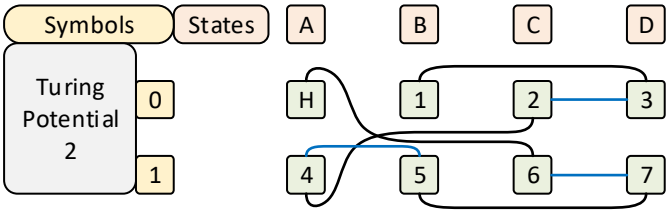


Figure 2.3

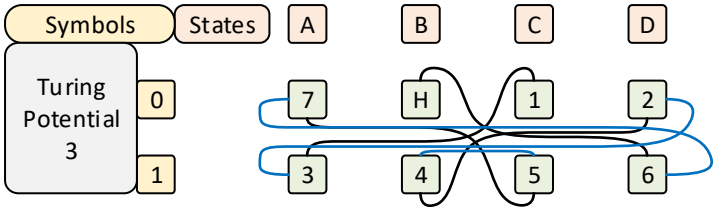


Figure 2.4

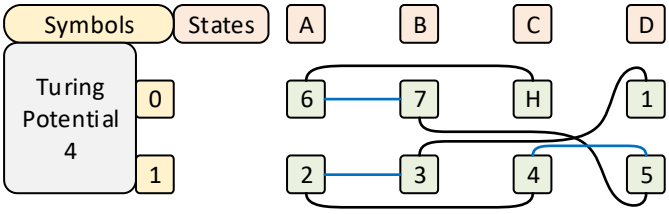


Figure 2.5

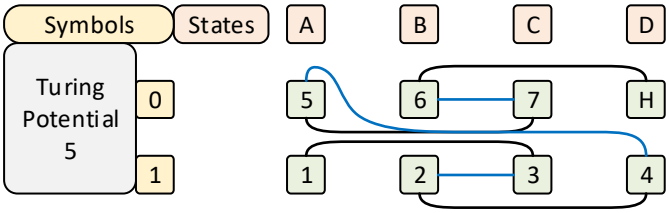


Figure 2.6

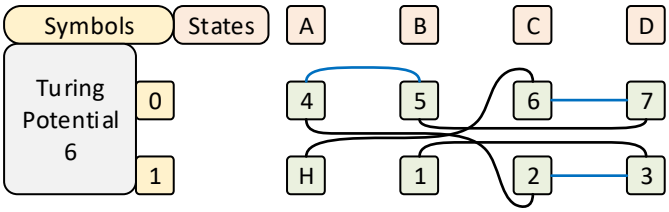


Figure 2.7

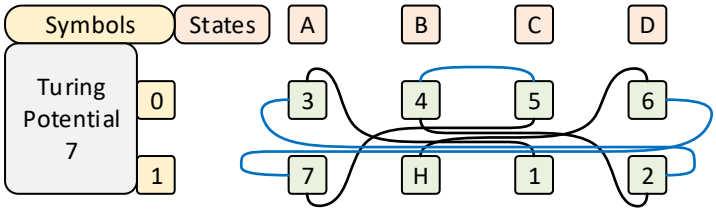


Figure 2.8

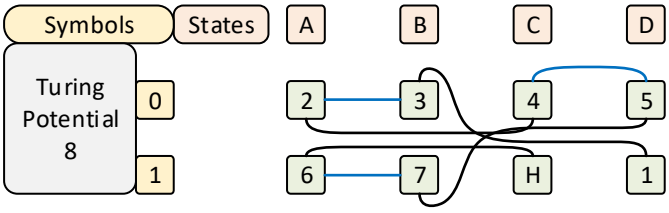


Figure 2.9

We are required to condense these potentials down as certain conditions must be met to allow for a state machine instruction set to be able to use all green nodes once.

Rules

A0 must be either 0 or 8 (H) - this allows to start at either the beginning of the tree or the end of the tree.

Each node (green node from the tree diagram) must be passed through once and only once.

NOTE: It is entirely possible to replace the H with a pointer to 1 and set a loop going to match the topology produced linking 1 to 8. The rules allow for this loop to be broken and the results inspected.

With these rules in place we narrow the selection to two diagrams.

Tree Set 1 State Machine Candidates

Potential 2

This can be selected initially on the merit of having H at A0, however, the connector being to 5 on the second row makes it not possible to build an instruction set that on the first step can get to symbol row 1.

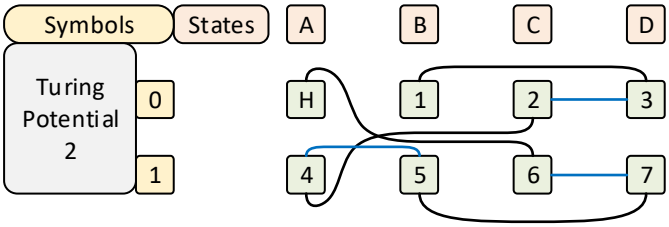


Figure 3.1

Potential 1

This is our state machine.

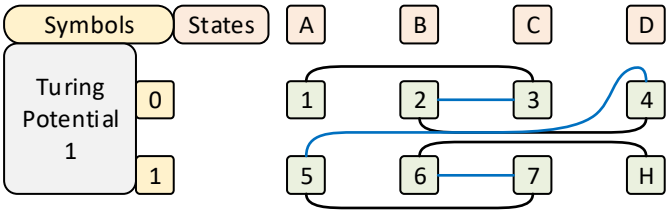


Figure 3.2

Inverting the State Machine

The exact inverse when flipped across the X and Y axis is represented in the same way.

The X axis is between symbols 0 and 1 row.

The Y axis is between B and C columns.

Consider that H = 1 and 1 = H (and each node numbering inverted in same way) in the inverted machine.

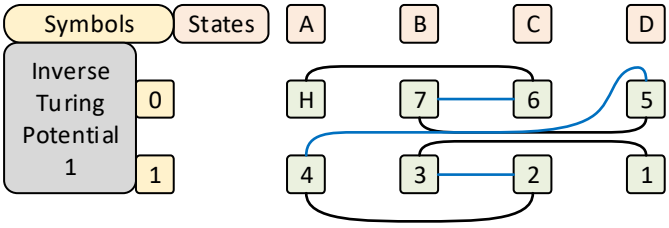


Figure 3.3

Turing Machine Construction

Consider the array pointer as unmoving. The array rotates left or right under the pointer head.

Can we find a state machine to satisfy the tree topology that we have as a candidate?

Yes.

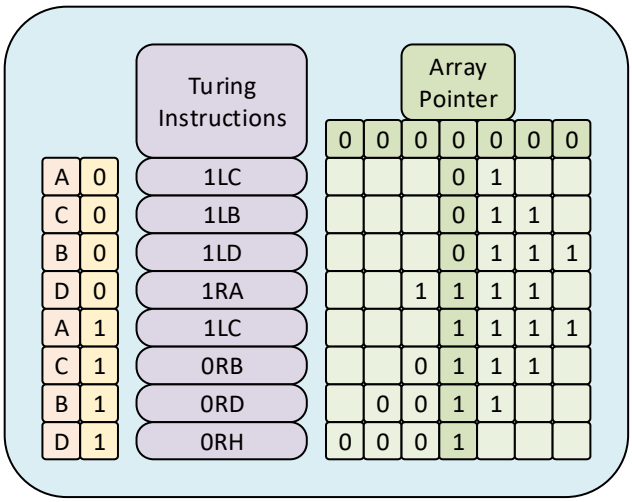


Figure 4.1

Now we know this, the inverse state machine is identical. We therefore have the answer to both state machines.

We can go further and create the inverse instruction set of the state machine. This is done simply by inverting left and right. The change in left or right shifting of the array results in the same pattern to each side of the pointer head.

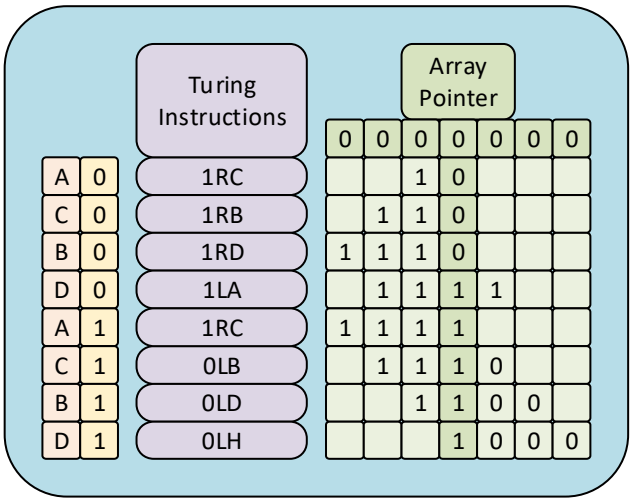


Figure 4.2

We now have 2 state machines, each that have 2 answers.

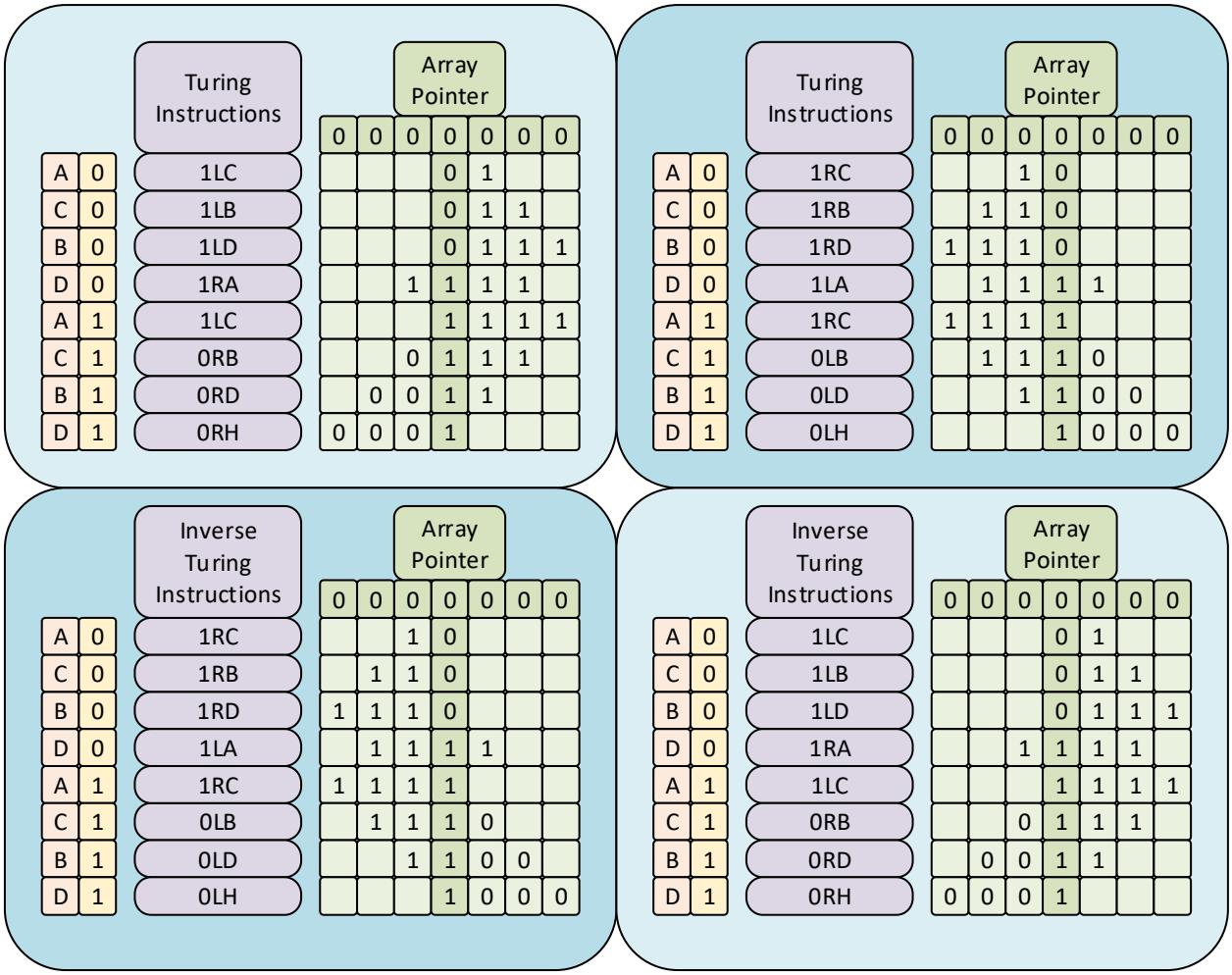


Figure 4.3

Let us represent the state machines with more proper notation:

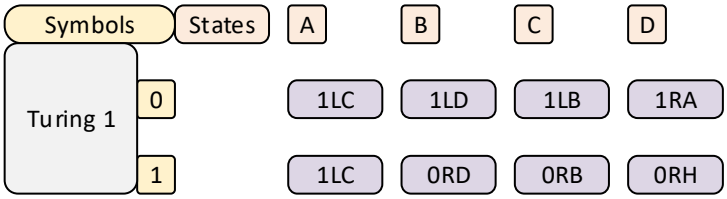


Figure 5.1

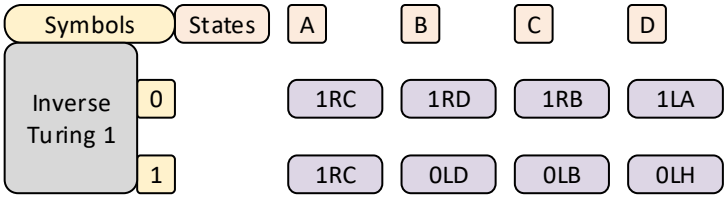


Figure 5.2

Combining the Array Matrix

We have 2 state machines and 4 results. The 2 state machines are the exact inverse of each other and achieve the same thing.

We have 4 answers to the same question effectively.

On combining them we must remember that one of the state machines is inverted and must be flipped along the X and Y axis before combining.

Combined Turing Machine 1 Results

0	0	1	0	0	1	0	0
0	1	1	0	0	1	1	0
1	1	1	0	0	1	1	1
0	1	1	2	2	1	1	0
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	0
0	0	1	1	1	1	0	0
0	0	0	1	1	0	0	0

Figure 6.1

Combined Inverse Turing Machine 1 Results

0	0	1	0	0	1	0	0
0	1	1	0	0	1	1	0
1	1	1	0	0	1	1	1
0	1	1	2	2	1	1	0
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	0
0	0	1	1	1	1	0	0
0	0	0	1	1	0	0	0

Figure 6.2

Flipped Inverse Turing Machine 1 Results = Turing Machine 1 (so we combine this into the first result as it is the same)

0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1
0	1	1	2	2	1	1	0
1	1	1	0	0	1	1	1
0	1	1	0	0	1	1	0
0	0	1	0	0	1	0	0

Figure 6.3

Result of combining the symbol array matrix:

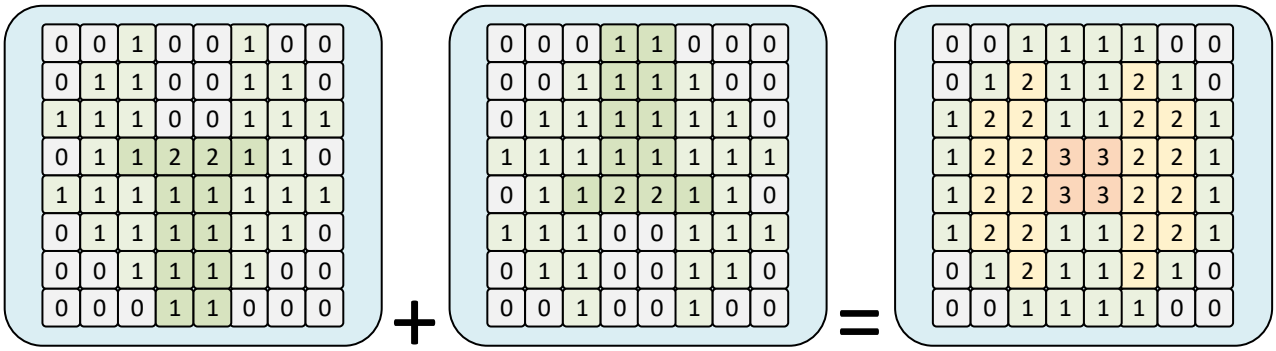


Figure 6.4

Result of negating the symbol array matrix with the un-flipped result from the inverse state machine:

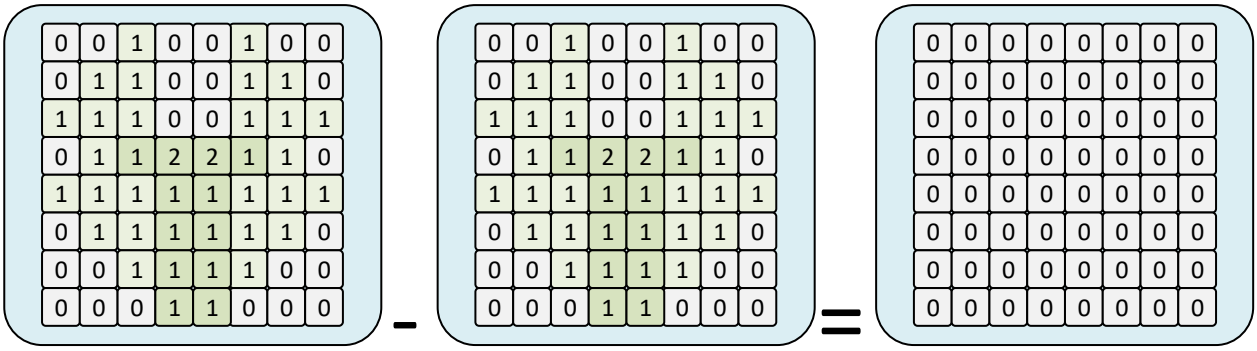


Figure 6.5

Building Blocks

The blocks have been arranged repeating pattern. There are no manipulations made to the rows in the matrix, it is as output by the state machine. D1 has been placed against D1. To repeat again place A0 against A0.

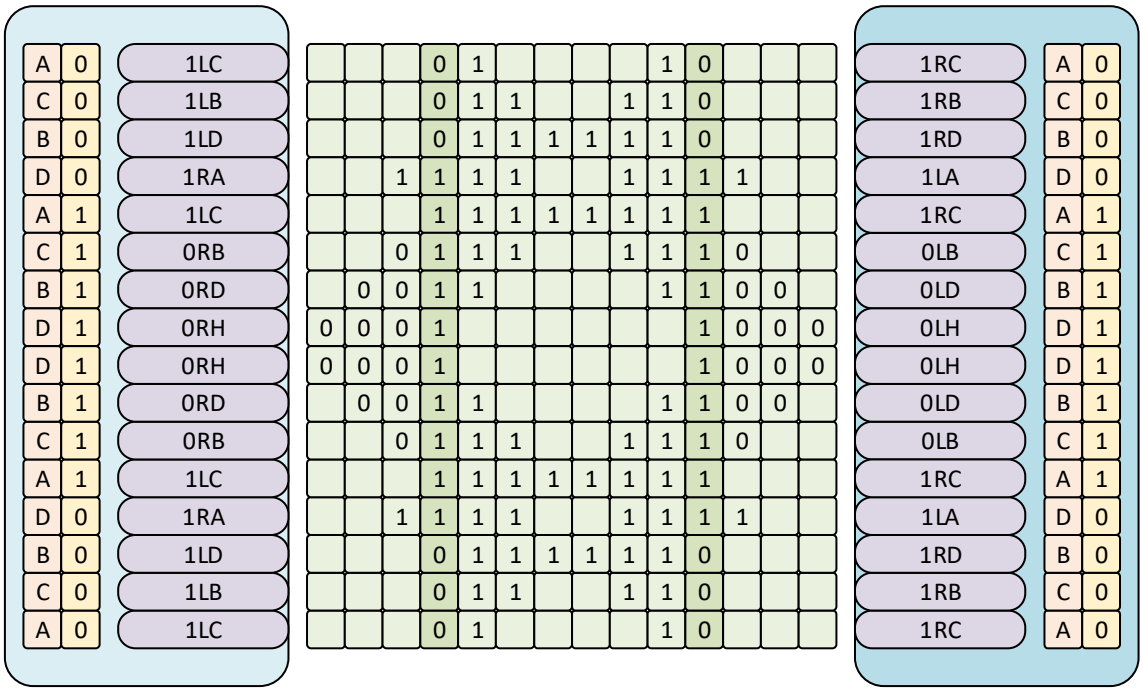


Figure 7.1

In the inverse output the pattern is the opposite due to no combining occurring yet.

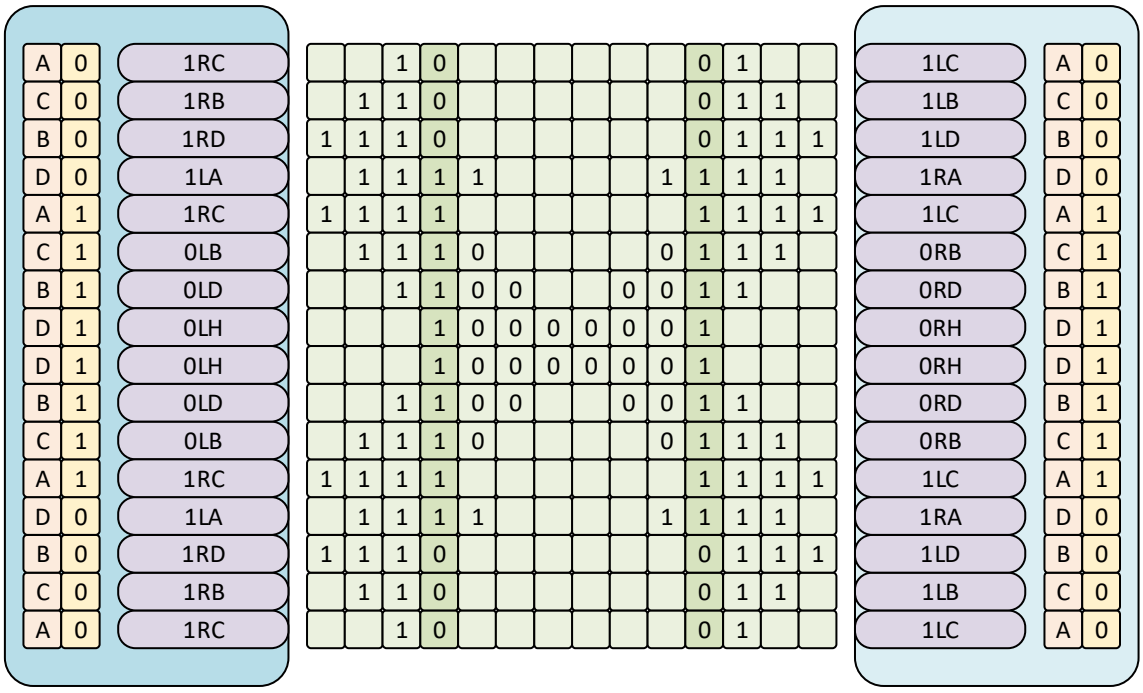


Figure 7.2

Stitching

The blocks can be stitched having been built together as above. It should be noted that both A0 and D1 end up being the same at either end of the blocks.

For this reason we could say they are equal and remove one of them, and so I do.

With a single D1, ie stitched together with D1 being the point through which the 2 matrix interact. NOTE: A0 would be removed at one end if stitching more repetitions of the block.

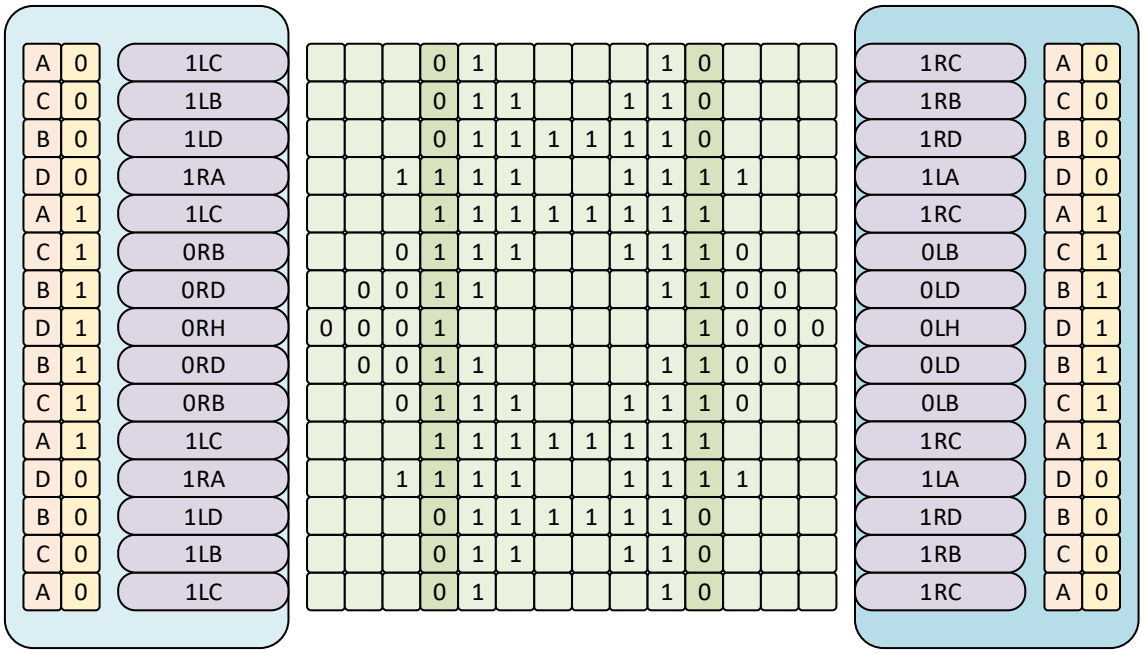


Figure 8.1

Again, the inverse with duplicated D1 removed. NOTE: A0 would be removed at one end if stitching more repetitions of the block.

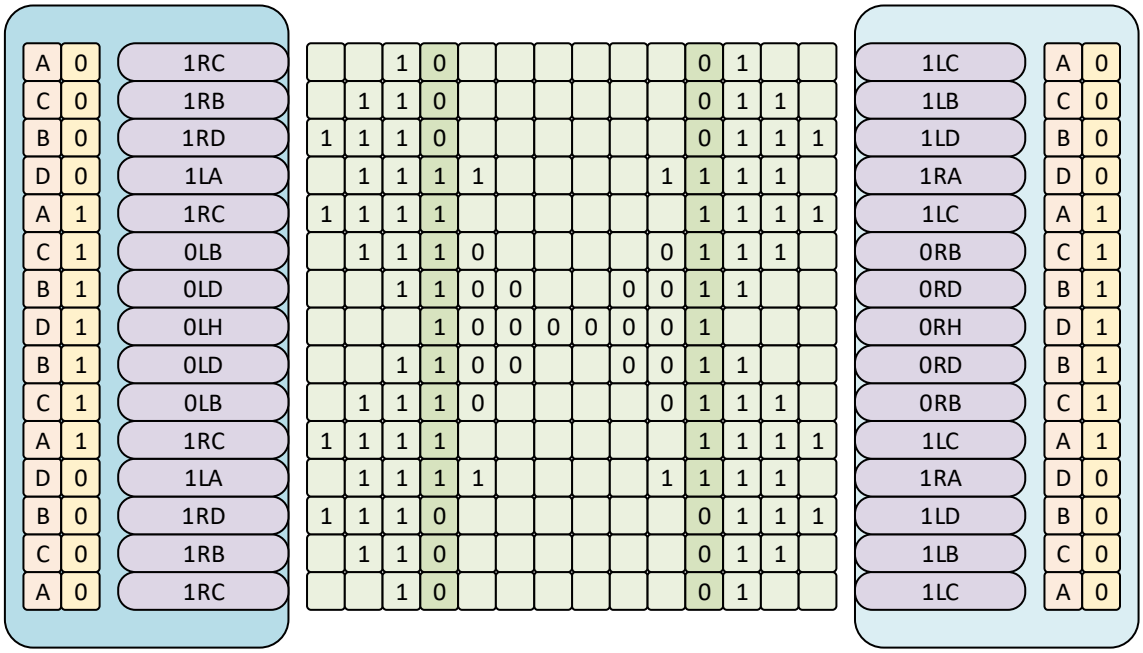


Figure 8.2

Stitching Analysis

Consider at this point we still have 2 array pointers that point at the green coloured tracks down through the pattern.

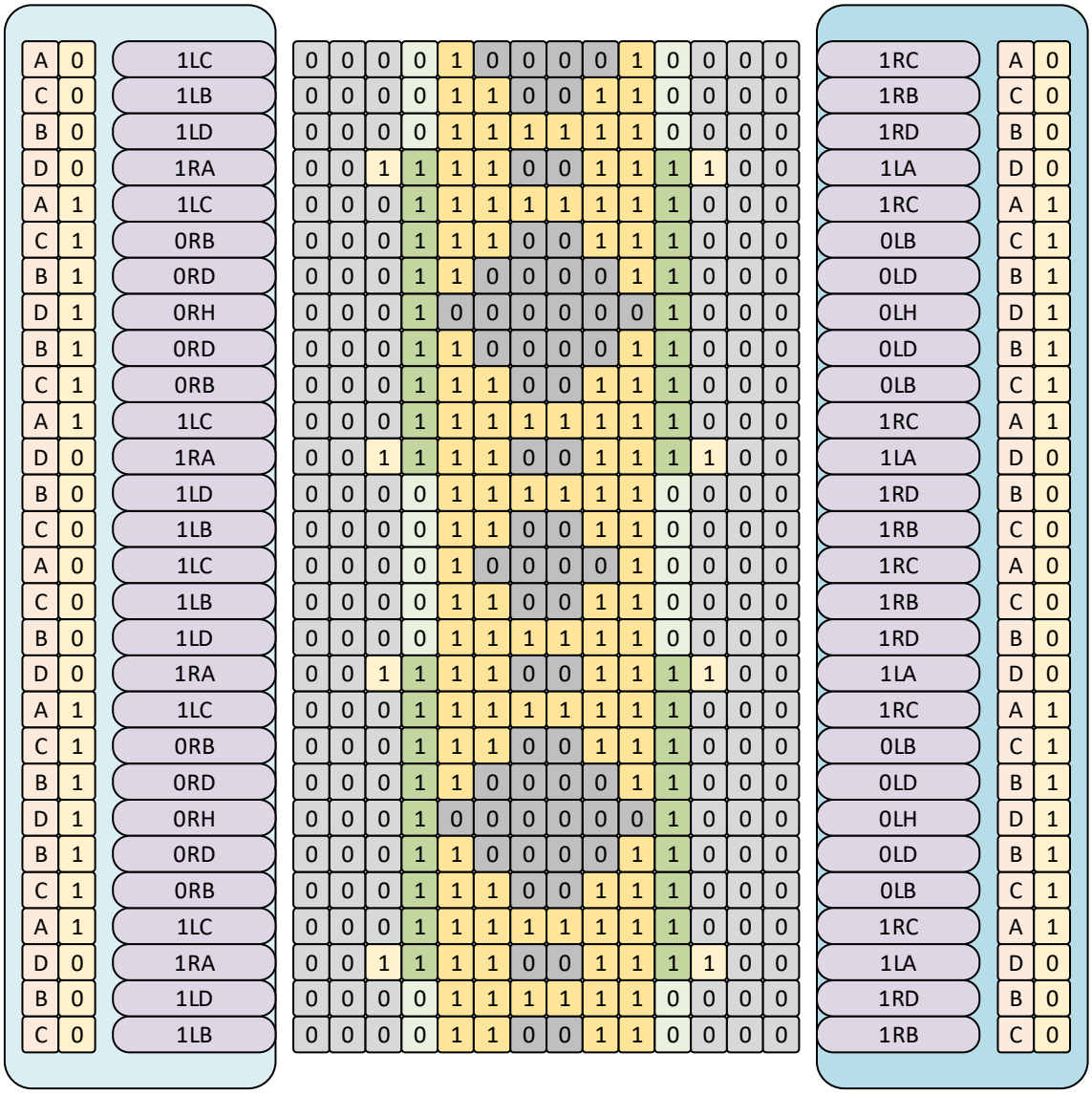


Figure 8.3

The inverse.

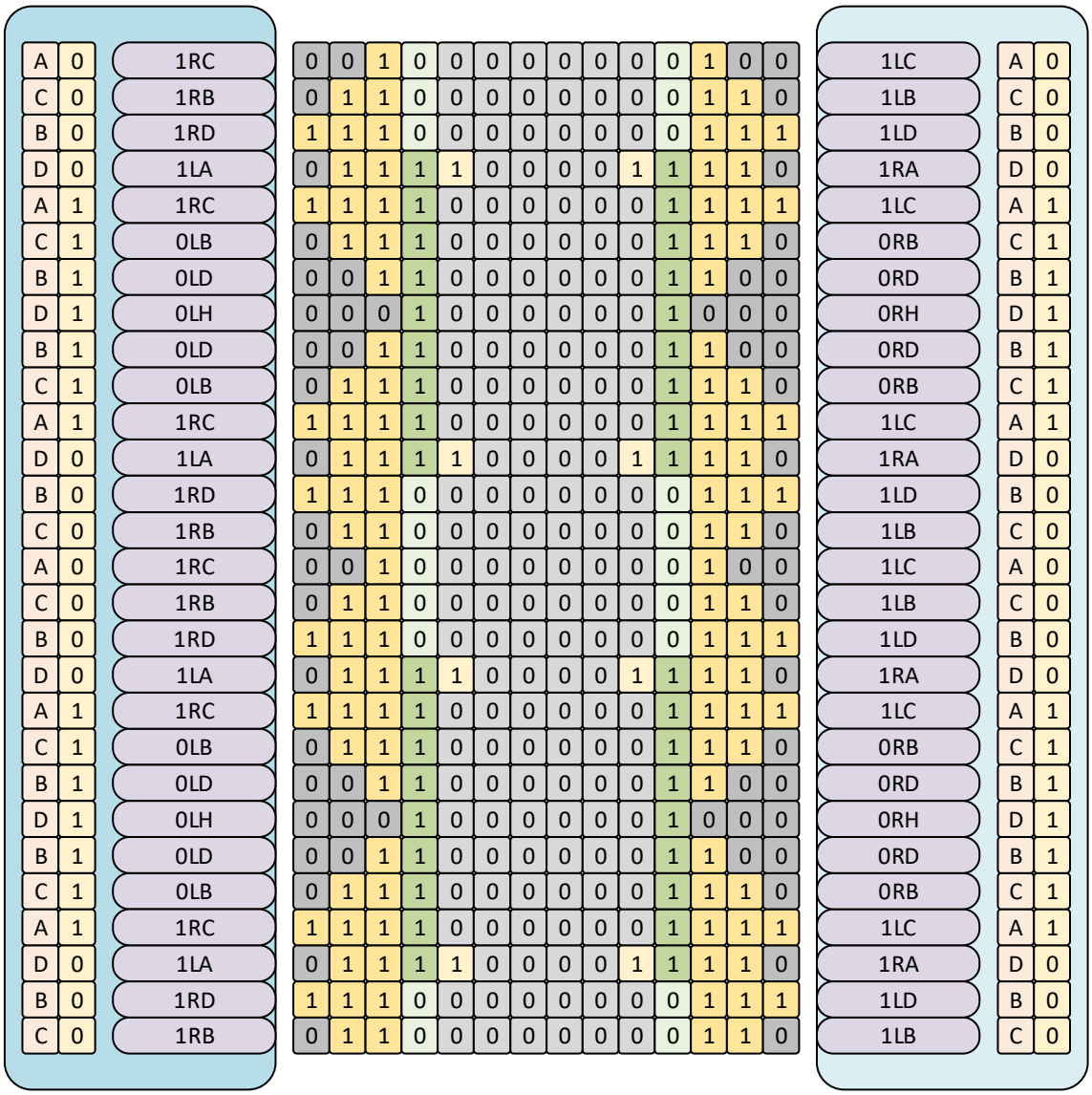


Figure 8.4

Combined Stitches

Let us bring back together the two separate matrices.

~~We still have 2 array pointer tracks and have only done one of the additions that would bring all 4 potentials together.~~

We still have 4 array pointer tracks to maintain an 8 by 8 symmetry. This seems to be indicated as the method to bring them together when applying this mechanism to matrices multiplication.

We are still required to flip X and Y on the inverse (right track) and combine it with the left track by folding this diagram down the Y axis.

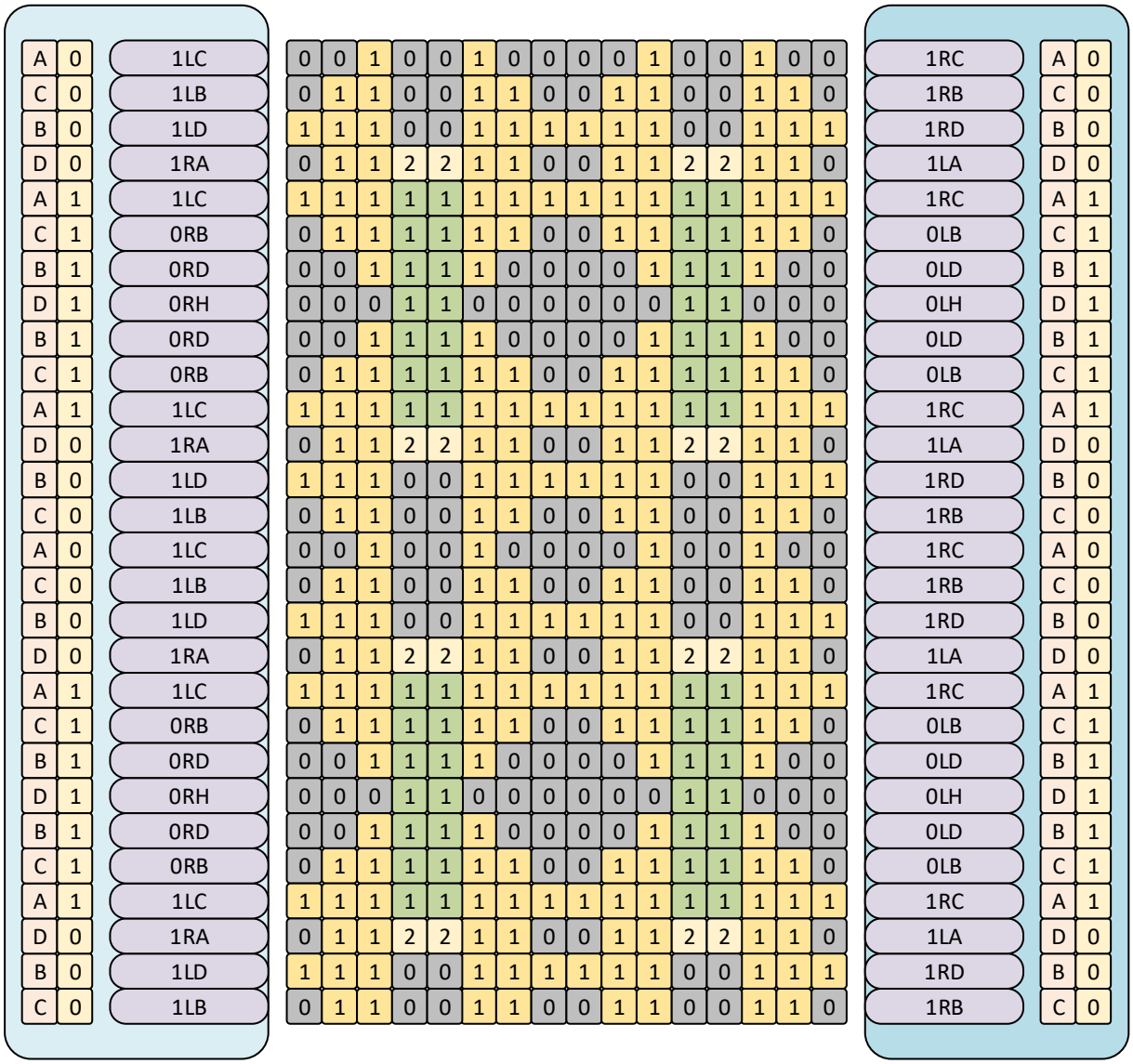


Figure 9.1

Combined Tracks

To invert the X and Y for the second track we can slide the track down so as A0 meets D1. Then we can fold across the Y axis of this diagram to combine the values of the two tracks. This is the same logic employed between figure 6.2 and 6.3 to un-invert the matrix.

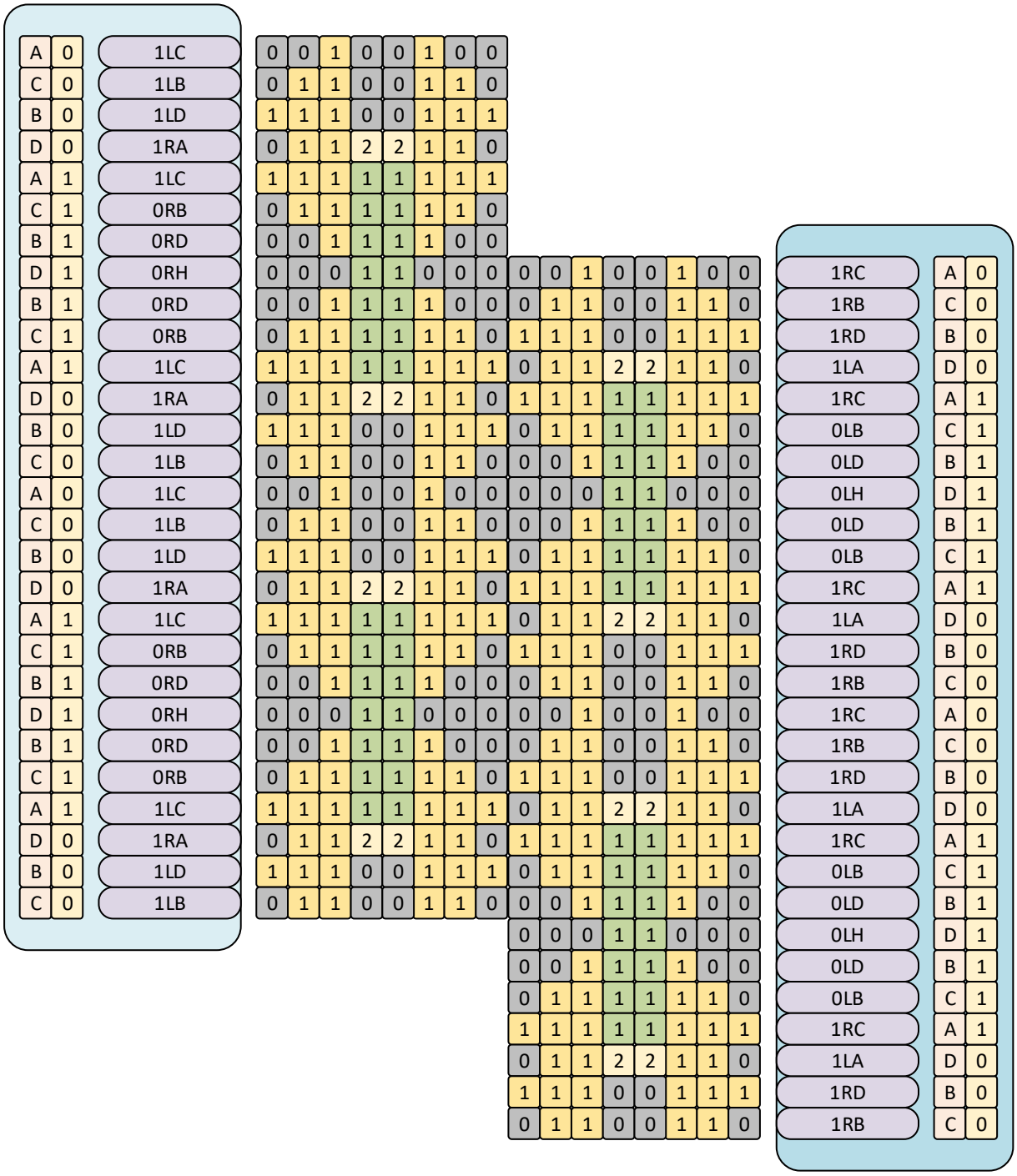


Figure 10.1

Result

There are a few things going on here, I will make some observations based solely on the logic that is used to build and get to this pattern.

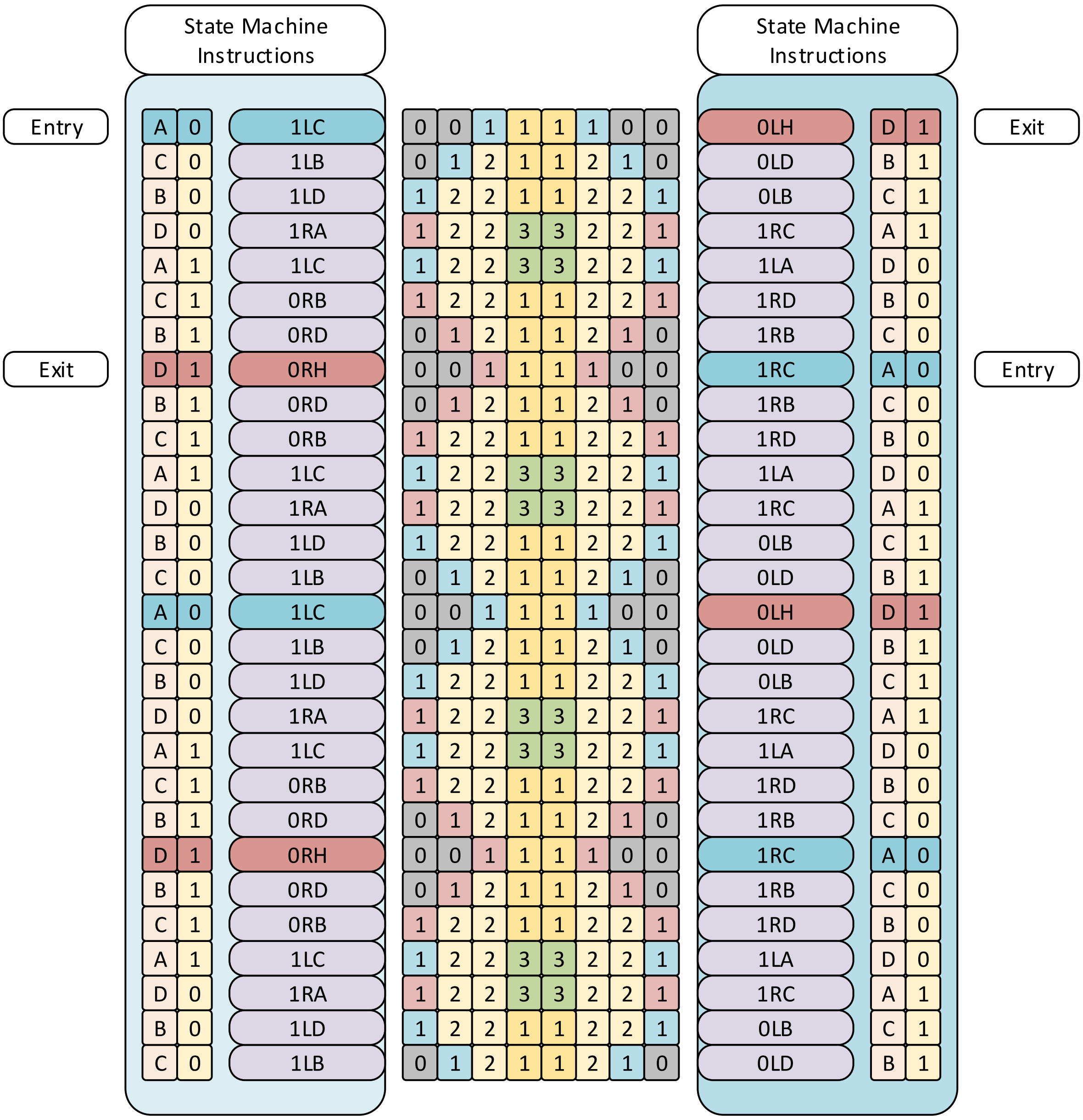


Figure 11.1

Observations

Observation #1

Red is the exit point for the state machine.

Blue is the entry point for the state machine.

Observations #2

If you know the instruction set for one half of the pattern, you know the instruction set for the other half.

Observation #3

On inspection of a row at any point through the pattern it is possible to know both the directly opposite and diagonally opposite direction of shift in the array.

For example:

C0 on the left = 1LB (this means the array will left shift)

Its direct opposite B1 on the right = 0LD (this means the array will left shift) we can work this out from B1 on our own instruction set being ORD.

Its diagonal opposite C0 on the right = 1RB (this means the array will right shift) we can work this out from C0 on our own instruction set being 1LB.

Observation #4

The alternating left and right shifts between A0 and D1 between left and right. There is an oscillation between A0 and D1. This in combination with oscillations within the instruction set itself creates an interesting interleave.

~~**Observation #5**~~

~~The interlocking nature of the 2's in the tracks before combination and the resultant manifestation of the only odd numbers due to the shift. This seems to occur due to the stitching, which would suggest a different result set having not removed the row, it would result in no odd numbers.~~

Observation #5 (Replacement)

Having produced: https://github.com/Sandcrawler/matrices

It suggests the tracks for the pointer head should not be combined to retain an 8 by 8 answer matrix. This provides an additional layer of symmetry.

Observation #6

If you consider the numbers in the result as height, the structure can be mapped in 3D. This can be done either by duplicating the pattern and placing it back-to-back or by evenly distributing the values through the Z plane.

Observation #7

It looks like, uncertain, that you could rotate the pattern 90 degrees and the 3D shape would interlock together well. It may interlock better by not employing stitching and just build the blocks.

Further Analysis

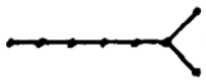
There is plenty of scope for additional analysis beyond what is in this paper using similar patterning.

Other Trees

I suspect that other trees will require a different twist on mapping into a state machine.

It may require some sort of recursion within the topology, instruction set or output array.

For example; Tree 2 from figure 10



2

Sigma(3,3)

I suspect that a similar topology can be mapped onto a sigma(3,3) state machine.

This would potentially allow for;

1) Looped result set between nodes 1-8 where node 9 (H) is never used.
2) Halting result set where node 9 is treated as an imaginary zero point outside of the loop.

If option 2 is used to build a pattern up, I suspect it may look similar to the pattern built with a sigma(4,2) however the resultant set when stitched could give 0 padding between the array matrices rather than being bound together as we see in the results for sigma(4,2) (??)

The array matrix results would yield larger numbers due to symbols being 0,1, and 2 and we are overlaying the matrices to combine them.