Awkward State Machines

Will Dengler

April 7, 2019

1 Introduction

Imagine you had a ball of yarn of infinite length and a pair of scizzors. Now choose some arbitrary length of yarn and cut it - we'll refer to this length as 1. Next, cut a second strand of yarn that has length 2 relative to your original piece of yarn and set it aside. Now, extend the yarn to length 3 and repeat the following instructions:

1. For every strand of yarn you have cut so far (other than the strand of length 1), check if the current extension of yarn can be split into even segments of the same length as the current strand by 'walking' the shorter strand up the longer one until you reach the end of or pass the end of the extended piece of yarn; if you reach the end of the extended piece exactly, then the extension can be divided into even segments. If none of the previous strands can be used to divide the current extension evenly, then double the length of the extended peice of yarn, then cut it in half, finally, set the cut strand aside with the others. Do not cut the extension if one of the previous strands does divide the extension evenly.

2. Using your strand of length 1, increase the length of the extension by 1.

3. Repeat the above two steps.

If you follow the above instructions, then order the strands of yarn you cut by their length, and finally wrote out their lengths relative to the strand of length 1, then you will find yourself writing down the prime numbers in consequetive order. The fact that this simple experiment derives the prime numbers using relative distance has always fascinated me. I've always had an itching notion that an algorithm based solely on relative distance, rather than integer arithmetic, would outperform the standard methods for discovering the primes in consequetive order. However, the problem of how to encode distance and divisibility without actually using numbers seemed to be impossible; after all, how do you tell a computer to use a ball of yarn and scizzors? And even if we could, the act of walking the strands up the extension is going to be pretty slow.

Let's modify our experiment slightly to include tacks and a corkboard. Start again by cutting some length of yarn for defining length 1. Now, using your piece of yarn, seperate two tacks on your board 1 unit away from one another. You can now determine the strand of length two by wrapping your ball of yarn around the tacks such that you start at one tack, wrap around the second, and then return to the first tack, then cut the yarn to produce the strand of length two. Now use the strand of length two to place two more tacks in your board at 2 units apart.Now using the tacks for length 1, create a strand of length 3 and repeat the following instructions:

1. For every pair of tacks on the board (other than the unit tacks), wrap the current extension of yarn around the tacks to check for divisibility. The extension is divisible if the yarn perfectly touches one of the tacks when it runs out of length. If none of the current pairs of tacks divide the extension evenly, then double the extension, cut it in half, and use the new strand to set apart a new pair of tacks on your board.

2. Use the unit pair of tacks to increase the length of the extension by 1.

3. Repeat the above two steps.

The above experiment is very simliar to the first. The pairs of tacks you've placed will wind up enumerating the primes once again. We are also still stuck with the problem of how could we encode this algorithm without integers, and the algorithm still isn't performing very quickly. However, the algorithm does highlight an important concept, our tacks use circles (technically ovals) in order to check the length of the extended piece of yarn. This is amazing because it allows us to check for divisiblity without counting anything out, instead, we just keep wrapping the yarn around until we get to the end; thus we don't need to know *how* many times we've wrapped the yarn around, we just have to look where it winds up at the end.

Let's modify our experiment once again. This time, you need tacks, a corkboard, and little flags that you can stick into the board (a needle with red tape for example). Start by placing two tacks down on the leftmost side of the board in a vertical line, and place a flag next to the bottom tack. Next, place three tacks in a vertical line to the right of those (imagine you are creating a bar graph with the tacks) and place a flag next to the highest tack. This expirement will require you to keep the vertical lines you create with the tacks distinct from one another, we shall refer to each line as a bar. Each bar in your board will always have a flag next to one of it's tacks. Repeat the following to run the experiment:

1. For every bar except the rightmost, move the flag to the tack above the one it is currently at; unless the flag is currently at the topmost tack in the bar, then move the flag to the bottom tack in the bar.

2. If there are no flags on any of the bottom tacks after you've moved every bar (other than the rightmost), then move the flag of the rightmost bar to its bottom tack and then create a new bar to the right of it that has one more tack than it, and place the new bar's flag at it's top tack.

3. If there was a flag on one of the bottom tacks, then add a new tack to the top of the rightmost bar, and place it's flag next to the new tack.

4. Repeat the above three steps.

If you run the above experiment and then count the number of tacks in each bar, you will find that the number of tacks in the bars enumerate the prime numbers in consequetive order. How did this happen? Well, this experiment is much more similar to the last two than it first appears. In this case, the way we move the flags on each bar is the same as wrapping the extended piece of yarn around the tacks in experiment two, or 'walking' the yarn up the extended piece of yarn from experiment one. However, this time our flags our able to preserve the state from the last iteration since you just have to move the flags one tack forward. The moving of each tack can be done in constant time for each bar, and is incredibly fast relative to the time it takes to re-wrap the extended strand of yarn around the tacks, or perform the 'walks'. Furthermore, this expirement doesn't need distance; rather, it only needs relative position (bottom tack, tack above that, ..., tack below top tack, top tack). By replacing distance with relative position, we can easily encode our experiment without the use of numbers using cycle graphs implemented via linked lists.

2 The Prime Graph Machine

3 Activation Cycle Machines

Definition
An activation cycle machine (ACM) is a directed cycle graph, C, containing $0 < n < |G|$ consequetive nodes called activation nodes.

We can refer to the ACM containing, a, activation nodes and, n, non-activation nodes as $C_{a,n}$.

An ACM's length is equal to the number of nodes it contains; thus, $C_{a,n}$'s length is $a + n$. We use the notation $|C|$ to refer to C's length.

The activation node that is connected to by a non-activation node is said to be at position 0, with all other nodes' positions being determined by their relative position to 0. Thus, there are $|C|$ unique positions the ASM C can have. We use the notation $\overline{C}$ to refer to C's position. Furthermore, we can identify the $|C|$ unique states as $C(0)$, $C(1)$, ..., $C(|C| - 1)$. We denote the set of unique states with $[C]$.

An ACM's initial state starts at position 0, then moves to the next consequetive position with each iteration. We can keep track of how which state we are on using the notation C^i, where i refers to how many iterations we've performed. Thus, C^0 denotes the initial state, C^1 the next, and C^{50} would be the 51st state. We call i the step of the ASM.

An ACM's state is said to be active if it's position is on one of it's activator nodes; otherwise, the ACM is called inactive.

Definition
For an ASM, C, we can define the $*$ operation between two states $C(i)$ and $C(j)$ as: $C(i) * C(j) = C(i + j \ mod \ |C|)$

Lemma
The $*$ operation for an ASM, C, forms a group with $C(0)$ as the identity. We can refer to this group as $< C >$.

Proof
TODO

Lemma
The group $< C >$ is isomorphic to the group of integers under addition modulo $|C|$.

Proof
TODO

Lemma
If $C_{a,n}$ is on position p at step m, then $m \ mod \ |C| = p$.

Proof
TODO

Corollary
C is active on step m if and only if $m \ mod \ |C| < n$

Definition
We can define the + operation between an ACM, C, and a positive integer n as: $C^i + n = C^{j+n}$.

Definition
We define the α operation from arbitrary state C^i to the unique state $C(i)$ as $\alpha(C^i) = C(i \ mod \ |C|)$.

Definition
Let $C^a < n >= \{ \ C(i) \mid C(i) = \alpha(C^a + jn)$ for some integer $j \ \}$

Lemma
$C^a < n >= [C]$ if $n \ mod \ |C|$ does not divide $|C|$.

4 Awkward State Machines

Definition
An Activation Branch is defined to be...

Definition
An Akward State Machine (ASM) is a machine for producing ACM's as it progresses through it's successive states. An ASM, $S_{a,n}$, begins with the ACM, $C_{a,n}$, and a activation branch of length $a+n+1$.Like an ACM, we can denote an ASM's state as S^i where i is the number of iterations performed to produce the state. In it's initial state, S^0, the position of the initial cycle C is 0.

To progress from state S^i to state S^{i+1}, increment the state each of the discovered cycle's by one. If none of the cycles are active after doing so, the machine copies the activation branch, then closes one of the branches to create a new ACM with a position of 0 (thus activating the new ACM). Finally, regardless of whether one of the previous cycles were active, add a new node to the end of the activation branch.

Definition
We'll define $[S^a] = \{\ C_{a.n}$ discovered by S on step $a\ \}$, and $[S] = \{\ C_{a,n}$ discoverable by $S\ \}$.

Lemma
For $S_{a,n}$, the length of the branch, B, on step k is given by $|B^k| = k+a+n+1$.

Proof
One step 0, the length of branch B is given by $|B^0| = 0 + a + n + 1$.
With each step, a single node is added to the branch B. Thus, $|B^{i+1}| = |B^i| + 1$.

Assume $|B^i| = i + a + n + 1$.
Then $|B^{i+1}| = |B^i| + 1 = (i + a + n + 1) + 1 = (i + 1) + a + n + 1$.

Thus we have shown that $|B^k| = k + a + n + 1$ by induction.
Q.E.D

Lemma
For $S_{a,n}$, if C_i is discovered on step k, then the length of $|C_i| = |B^{k-1}| = k + a + n$.

Assume for $S_{a,n}$, that C_i is discovered on step k.

To produce state k, the ASM algorithm first moved all C_j for $j < i$ from state C^{k-1} to state C^k. After doing so, there did not exist a $j < i$ such that C_j was active. Therefor, the algorithm copied the branch B^{k-1} and closed one of the two branches to create C_i.

Thus, the length of C_i is equal to the length of the branch B^{k-1}:
$|C_i| = |B^{k-1}| = (k - 1) + a + n + 1 = k + a + n$.
Q.E.D

Lemma
For $S_{a,n}$, $|C_i| \geq |C_{i-1}| + a$.

Proof
Assume cycle C_i is discovered by $S_{a,n}$ on step k.

Then $\overline{C_i^k} = 0$.
Furthermore, the new branch, B, will have $|C_i| + 1$ nodes at step k.

For next, $1 \leq j < a$ steps, the cycle C_i^{k+j} will be active since there are a activation nodes.
Furthermore, the length of the branch B at each step will be given by $|B| = |C_i| + j + 1$.

The ath step after discovering C_i will be the first time that C_i will be inactive. The branch length of the $(a-1)$th step is given by $|B^{j+a-1}| = |C_i| + a$. Thus, if C_p for $0 \leq p < i$ are also inactive on step ath step after discovering C_i, then we will have to close B on the ath step, thus creating cycle C_{i+1} with length $|C_{i+1}| = |B^{j+a-1}| = |C_i| + a$. Furthermore, if any of the cycles C_p were active on the ath step after discovering C_i, then the branch would not close, thus the next cycle's length is at least as long as the branch on the $(j+a)$th step: $|C_{i+1}| >= |B^{j+a}| = |C_i| + a + 1$.

Thus, we have shown that $|C_i| \geq |C_{i-1}| + a$.
Q.E.D

Lemma
For any $C_i, C_j \in [S_{a,n}]$ with $j < i$, it holds that $|C_i| \geq |C_j| + (i-j)a$.

Proof
If $i = j+1$, then $|C_i| \geq |C_j| + a = |C_j| + (i - i + 1)a = |C_j| + (i+j)a$ by a previous lemma.

By the same token of logic, $|C_i| \geq |C_{i-1}| + a \geq (|C_{i-2}| + a) + a = |C_{i=2}| + 2a \geq (|C_{i-3}| + a) + 2a = |C_{i-3}| + 3a \geq ... \geq |C_{i-(i-j)}| + (i-j)a = |C_j| + (i-j)a$.

More formally, assume for $1 \leq k - i$ that $|C_k| \geq |C_i| + (k-i)a$.
Then $|C_{k+1}| \geq |C_k| + a \geq (|C_i| + (k-i)a) + a = |C_i| + (k+1-i)a$.

Thus we have shown that $|C_i| \geq |C_j| + (i-j)a$ by induction.
Q.E.D

Lemma
For any step, k, for any $C_i \in [S_{a,n}^k]$, it holds that $\overline{C_i^k} = (k + a + n) \ mod \ |C_i|$.

Proof
The initial cycle C_0 starts at position 0 on step 0. By properties of ACM's, we know that the position of C_0 is given by:

$\overline{C_0^k} = k \ mod \ |C_0| = [k + (a+n)] \ mod \ (a+n) = (k+a+n) \ mod \ |C_0|$.

If a cycle, C_i, is discovered on step k, then we know it's length is given by $|C_i| = k + a + n$. Furthermore, when it's dicovered, it's position is set to 0.

Thus, the position of C_i^k is given by:

$\overline{C_i^k} = 0 = (k+a+n) \ mod \ (k+a+n) = (k+a+n) \ mod \ |C_i|$.

By properties of ACM's, the position for all steps $j > k$ will be given by:

$\overline{C^j} = (\overline{C^k} + (j-k)) \ mod \ |C^i| = (0 + (j-k)) \ mod \ (k+a+n)$
$= (k+a+n) + (j-k) \ mod \ |C_i| = (j+a+n) \ mod \ |C_i|$.

Therefor we have shown that the position of C_i on step k for any $C_i \in [S_{a,n}^k]$ is given by $\overline{C_i^k} = (k+a+n) \ mod \ |C_i|$ for any C_i in $[S^k]$.
Q.E.D

Lemma
For any $C_i, C_j \in [S_{a,n}]$ with $j < i$, it holds that $|C_i| \ mod \ |C_j| \geq a$.

Proof
Assume for ASM, $S_{a,n}$, that ACM, C_i, is discovered on step k.

Then $|C_i| = k + a + n$.
We know also know that the position of C_j^k must be greater than or equal to a; otherwise, C_j would be active on step k and we would not have discovered C_j.

Furthermore, we know the position of C_j^k is given by:
$\overline{C_j^k} = (k+a+n) \ mod \ |C_j|$.

Finally, subsitituting gives us: $\overline{C_j^k} = |C_i| \ mod \ |C_j| \geq a$.

Thus we have shown that, $|C_i| \ mod \ |C_j| \geq a$.
Q.E.D

Lemma
$S_{1,1}$ produces the prime numbers.

Lemma
If $C_i, C_j \in [S]$ with $i < j$, then $C_j < |C_i| >= [C_j]$

Proof
TODO

Theorem
For every ASM, S, the set $[S]$ is non-finite.

Proof
Assume there exists an ASM, $S_{a,n}$ such that $[S_{a,n}]$ is finite.
Then there must exist some integer, j, such that $\forall k \geq j$, $[S^k] = [S]$.

Move to the next state S^k from S^j such that the initial cycle, C_0, is at position a.

Now repeatedly move S $|C_0|$ states at a time until C_1 is at position a. Notice that by moving $|C_0|$ positions at a time, $\overline{C_0} = a$ as well.

Now repeatedly move S $|C_0| * |C_1|$ state's at a time until C_2 is at position n. Notice that both C_0 and C_1 are also still on position a (view it as moving $|C_0|$ moves $|C_1|$ times and as moving $|C_1|$ moves $|C_0|$ times).

Continue to repeat this process in order until all the cycles in $[S]$ are at position a. Explicity, when cycle's C_0, C_1, ..., C_i are all at position a, move $|C_0| * |C_1| * ... * |C_i|$ positions at a time until cycle C_{i+1} is at position a as well.

Once all cycles are on position a, move the S to the next state. Then all the cycles will be at position $n + 1$ which is not an activator node. Thus, we would need to add a new cycle to the ASM. Therefor, $[S^j] \neq [S]$ for any integer j by contradiction. Thus, $[S]$ is non-finite for all ASM's.
Q.E.D

Corollary
There are an infinite number of prime numbers.

Proof

$S_{1,1}$ produces the prime numbers, and $[S_{1,1}]$ is non-finite by the above theorem.
Q.E.D