Awkward State Machines

Will Dengler

July 24, 2020

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 Cycle Graphs

In order for us to study awkward state machines in any detail, we are going to need to understand their underlying components. As such, let us start by examining the simplest component in an awkward state machine: the cycle graph.

Definition
A *cycle graph* is a directed, connected graph whose points form a circle. More explicity, for any $n > 1$, the cycle graph G_n has points:

$$P = \{ \ p_i \mid 0 \leq i < n \ \}$$

such that for any $i < n - 1$, p_i has only one edge which goes to p_{i+1}, and p_{n-1} has only one edge going to p_0.

Our definition tells us that a cycle graph is nothing more than a circular, directed graph with an explicit and straightforward labeling of its points. Our study of cycle graphs will be predominantly focused on the outcomes of walks about cycle graphs. As such, for the sake of clarity in our writing, let us now take the time to define two functions - the *walk function* ω and the *index walk function* Ω - that we can use to talk about walks of arbitrary length around any cycle graph.

Definition
For any cycle graph G_n with points P, we define the *walk function* $\omega : (P, \mathbb{N}) \to P$ to be $\omega(p_i, k) = p_j$, where p_j is the point arrived at after a walk of length k when starting at point p_i on cycle graph G_n.

Furthermore, we define the *index walk function*, $\Omega : (P, \mathbb{N}) \to \mathbb{N}$ to be $\Omega(p_i, k) = j$, where j is the index of the point $\omega(p_i, k)$.

We already know a little bit about the walk and walk index functions from our definition of a cycle graph. Our defition tells us point that p_{n-1} only has one edge going to point p_0. Furthermore, for any other point p_k there is only a single edge going to point p_{k+1}. Thus, we have:

$$\omega(p_{n-1}, 1) = p_0 \text{ and } \Omega(p_{n-1}, 1) = 0$$
$$\omega(p_k, 1) = p_{k+1} \text{ and } \Omega(p_k, 1) = k + 1 \text{ for any } k < n - 1$$

In order for our mappings ω and Ω are to be valid for walks of arbitrary length, then it must be the case that for any length, k, and any point, p_i, within a cycle graph, there must be only a single walk that can be taken of length k that starts at point p_i. As such, we shall begin our study by proving just this.

Lemma
For any cycle graph G_n, for any point p_i within that graph, there is only a single walk of length k that begins at point p_i.

In terms of our walk functions, this lemma states that ω and Ω are defined for any point p_i and any walk length k. Furthermore, $\omega(p_i, k) = \omega(p_j, k)$ and $\Omega(p_i, k) = \Omega(p_j, k)$ if and only if $p_i = p_j$.

More simply stated, ω and Ω are valid mappings.

Proof
The truth to this lemma is result of the fact that every point in a cycle graph only has a single edge. As such, we shall use this fact to complete a proof by induction on the length of our walk.

Base Case
Our base case, a walk of length one starting from any point p_i, is obviously true since there is only a single edge that can be traversed from any point. In other words, for any point p_i, we know that $\omega(p_i, 1)$ is defined.

In order to prove the uniqueness aspect of our base case, we shall assume that $\omega(p_i, 1) = \omega(p_j, 1)$ for some points p_i and p_j and then show that p_i and p_j are actually the same point. In other words, if a walk of length one starting at point p_i ends at the same point that a walk of length one from point p_j does, then it must be the case that our starting points p_i and p_j are in fact the same point.

Let us begin by examing the case where our walk ends at the initial point p_0. In other words, let us assume $\omega(p_i, 1) = \omega(p_j, 1) = p_0$. We already know that $\omega(p_{n-1}, 1) = p_0$ by definition of our cycle graph, so now all we have to do is rule out the possibility that for any other point p_k, where $k < n - 1$, that $\omega(p_k, 1) \neq p_0$. But we already know this to be true since $\Omega(p_k, 1) = k+1 > 0$ for all $k < n - 1$.

Now let us wrap our our uniqueness by assuming $\Omega(p_i, 1) = \Omega(p_j, 1) = k$ for some $k > 0$, and showing that this results in $p_i = p_j$. By the definition of a cycle graph, we know that point p_{k-1} has only a single edge which goes to point p_k, thus $\omega(p_{k-1}, 1) = p_k$. Furthermore, for any $l < k-1$, $\Omega(p_l, 1) = l + 1 \leq k - 1 < k$. Thus, it must be the case that p_{k-1} is the only point that allows us to take a walk of length one to arrive at point p_k.

Hypothesis
Assume that for any $k > 1$ and any point p_i, that there is only a single walk of length $j < k$ that can be taken when starting at point p_i. In other words, $\omega(p_i, j)$ is valid whenever $j < k$.

Inductive Step
If we wish to take a walk of length k starting at any point p_i in our graph, we can do so by first taking a walk of length $k-1$, and then extend that walk by one more step.

By our inductive hypothesis, we know that there is only a single walk of length $k-1$ we can take starting at point p_i. Furthermore, if we were to extend that walk by one more step, our base case tells us that there is only a single walk that can be taken from ending point $\omega(p_i,\, k-1)$ of length one. Thus, there can only be a single walk of length k since it is a composite of our two shorter walks; furthermore, we have shown:

$$\omega(p_i, k) = \omega(\omega(p_i,\, k-1),\, 1)$$

Now that we know that our walks starting from any point of any length are not only valid, but also unique with regard to the starting point and length, we shall turn our attention to the predicting which point our walks will end on. To start, we shall examine the shortest walk between any two point p_j and p_k where $j < k$.

Lemma
For any cycle graph G_n, for any $j < k < n$, the shortest possible walk from point p_j to p_k has length $k - j$.

Expressed via the walk function, we have that:

$$\omega(p_j, k - j) = p_k \text{ whenever } j < k$$

Furthermore,

$$\omega(p_j, l) \neq p_k \text{ whenever } l < k - j$$

Proof

We shall note a few things about cycle graphs before we begin. First, for any point $p_j \in G_n$, there is only one edge that can be traversed. As such, there is only a single walk that can be made from p_j of length k, for any k. Secondly, since we've labeled our points such that point t's only edge goes to point $t + 1$ for $t < n - 1$; then for any $j < k < n$, we can think of a walk from p_j to p_k as a walk down the number line, and as such, the shortest walk from point p_j to p_k will be of length $k - j$.

Before we get swept up into the theory of cycle graphs, let us first note some properties of our newly defined functions, ω and Ω

First off, we acknowledge that ω is a valid mapping there is only a single walk of length k, for any k, that we can take starting at any point since each of our points only have a single edge they can traverse.

By definition of a cycle graph, point p_{n-1} only has one edge going to point p_0. Furthermore, for any other point p_k there is only a single edge going to point p_{k+1}. Thus, we have:

$$\omega(p_{n-1}, 1) = p_0 \text{ and } \Omega(p_{n-1}, 1) = 0$$
$$\omega(p_k, 1) = p_{k+1} \text{ and } \Omega(p_k, 1) = k + 1 \text{ for any } k < n - 1$$

Furthermore, we know that if we want to take a walk of length k, for any $k > 1$, starting from any point, p_i, then this can be accomplished by first taking a walk of length g, where $1 \leq g < k$, starting at point p_i to arrive at some point p_j; and then we can extend our walk by taking a second walk of length $k - g$ starting at point p_j. Applying this to the walk function, we get:

$$\omega(p_i, k) = \omega(\omega(p_i, j),\ k - j) \text{ for any } 0 < j < k.$$

Now we are ready to dive into the theory of cycle graphs. We shall begin our study by examing the shortest walk to the point p_0 from any point within the cycle graph.

Lemma
For any cycle graph G_n and for any point p_i within that graph, a walk of length $n - i$ will end at point p_0. Furthermore, this the shortest walk between any point p_i and p_0.

Applying the lemma to the walk function, we get: $\omega(p_i, n - i) = p_0$ for every point p_i; furthermore, $\omega(p_i, j) \neq p_0$ whenever $j < n - i$.

Proof
We shall break our proof into two cases. We shall first consider the walk from the last point p_{n-1} to the initial point p_0. Thus, our second case shall consider the walk to the initial point for all other points in our graph.

Case 1
First, let us consider the trivial cases where $j = n - 1$. By definition of a cycle graph, the point p_{n-1} has only one edge going to point p_0, therefor, our walk has length $1 = (n - n) + 1 = n - (n - 1)$. Furthermore, a walk of length 1 is the shortest possible walk of any graph, thus our walk from p_{n-1} to p_0 must be the shortest walk possible between these two points.

Now, let us consider the walk from p_j to the point p_{n-1} for $j < n - 1$.

As we noted above, the shortest walk between any two points $a < b < n$ will be of length $b - a$ by virtue of our labeling of the points of a cycle graph. As such, the shortest walk from p_j to p_{n-1} will have length $(n - 1) - j$.

By definition of a cycle graph, we know that point p_{n-1}'s only edge goes to point p_0. As such, if we extend our walk from point p_j to p_{n-1} one step further, we will have arrived at point p_0, thus giving us a length of $((n - 1) - j) + 1 = n - j$.

Furthermore, our walk from p_j to p_{n-1} was the shortest possible walk from p_j and p_{n-1}; and our extension of that walk from p_{n-1} to p_0 was also the shortest possible walk from p_{n-1} to p_0. As such, our composite walk from p_j to p_0 must be the shortest possbile walk from p_j to p_0.

We shall take a moment to note that as a direct result of the above lemma, the shortest walk from point p_0 back to point p_0 will have length $n - 0 = n$. This matches our intuition nicely since a walk from p_0 back to itself is similar to a lap around a race track, and as such will have to visit every point in our cycle graph, and we have n total points.

In our next lemma, we will see that a walk of length n from any point in our graph will return us back to the point whence our walk originated. This should be fairly obvious since for any point p_j, we could relabel our graph to make p_j our initial point p_0, and we already know that the shortest walk from p_0 back to itself has length n.

Lemma
For any cycle graph CG_n, a walk of length n from p_i will end at point p_i. Furthermore, this is the shortest possible walk from any point back to itself.

Proof
As we saw in the previous lemma, for any point $p_i \in CG_n$, a walk of length $n - i$ is the shortest possible walk from p_i to point p_0.

Furthermore, we also know that the shortest walk from p_0 to any point p_j for $j > 0$ will have length $j - 0 = j$.

Combining these two facts together, we can take a walk from p_i back to p_i by first taking a walk to p_0 in $n - i$ steps, and then continuing our walk from p_0 to p_i in i more steps, giving us a total walk length of $(n - i) + i = n$.

Furthermore, our constructed walk is composed of two shortest possible walks, and is therefor the shortest possible walk from p_i back to itself.

Now that we know we can return to point p_i by taking a walk of length n around our graph, we can easily extend this idea to show that a walk of length kn for $k > 0$ will also return us to point p_i. As such, a walk of length kn is equivelant to taking k laps around our cycle graph.

Corollary
For any cycle graph CG_n, a walk of length kn, $k > 0$, from p_i will end at point p_i.

Proof
Let us prove this corollary via induction on k, the number of laps around our graph.

The base case, $k = 1$, is the result of our previous lemma, in particular, that a walk of length $n = 1n =$ will return us back to point p_i.

Now, let us assume that for all $j < k$, that a walk of length jn from point p_i will return us to point p_i.

By assumption, we can take a walk of length jn starting at point p_i to arrive back at point p_i. As such, if we extend this walk by n more steps, we will again return to p_i by the above lemma; thus giving us a total walk of length $jn + n = (j + 1)n$.

With the proof of the above lemma, we now have enough basic knowledge about cycle graphs in order to reason about the result of any walk on our graph. Furthermore, we shall see that our cycle graphs, and the walks we take around them, are merely another way of thinking about modular arithmetic.

Before we state our main theorem for cycle graphs, let us get some notation out of the way. For non-zero integer, n, and any integer $j > 0$, we write $j \ mod \ n = k$ (read "j modulo n is k") to imply that $j = mn + k$ where $0 \leq k < n$, for some integer m.

Theorem
For any cycle graph C_n, for any $k > 0$, for any $p_i \in C_n$, a walk of length k starting at p_i will end at point p_j, where $j = (i + k) \ mod \ n$.

Proof
To began our proof, let $k > 0$ be the length of our walk of arbitrary length around our cycle graph, CG_n, starting from any point p_i.

Rather than trying to prove that the statement is true for arbitrary k, we shall instead break our proof into four seperate cases:

1. When our walk is short enough to not take us past point p_{n-1}: $k < n-i$

2. When our walk ends at point p_0: $k = n - i$

3. When our walk takes us past point p_0, but is no longer than the number of points in our graph: $n - i < k \leq n$

4. When our walk is longer than the number of points in our graph: $n < k$

For each of these cases, we shall show that we will end our walk on some point, p_j, such that $0 \leq j < n$; and $i + k = j + xn$, for some integer x. In doing so, we will have proven that $j = (i + k)\ mod\ n$ by the definition of modular arithmetic.

Case 1
Let us first consider the case where our walk is short enough to not take us past point p_{n-1}, in other words, let us consider the case where $k < n - i$.

In this case, we know our walk will end at some point p_j, where $j = i + k$. Since we have $k < n - i$, we also know that $0 < i + k = j < n$. Furthermore, $i + k = j = j + 0n$, and as such, $j = (i + k)\ mod\ n$.

Case 2
Now let us consider the simple case where $k = n - i$. By a previous lemma, we know that a walk of lenth $n - i$ will end on point p_0 if started from point p_i. Furthermore, $i + k = i + (n - i) = n = 1n + 0$, thus, $0 = (i + k)\ mod\ n$.

Case 3
Now let us consider a slightly longer walk, but is still no longer than the number of points in our graph. In other words, let us consider a walk of length $n - i < k \leq n$.

We know that the first $n - i$ steps of our walk will end at point p_0, leaving us with a remaining of $j = k - (n - i)$ steps in our walk. Furthermore, we have:

$$(n - i) - (n - i) = 0 < k - (n - i) = j \leq n - (n - i) = i < n$$

Since $j < n$, we know that extending our walk j more steps from point p_0 will end on point p_j to complete our walk of length k. Not only that, but we also have

$$i + k = k + i + (n - n) = k - n + i + n = k - (n - i) + n = j + n$$

Thus, we know that $j = (i + k)\ mod\ n$.

Case 4
Now let us turn to the final case we must consider, the case where our walk length is greater than the number of points in our graph; in other words, let $k > n$. To complete our proof, we shall show that we can reduce this final case to one of our previous three cases.

Since k is a positive integer, we can find a positive another integer a such that $an < k \leq (a+1)n$. As such, by our previous corollary, we know that we can walk the first an steps around our graph to return us to point p_i, and leave us with a remainder of $l = k - an$ steps in our walk of length k.

Since we have returned to point p_i with $0 < l \leq n$ steps remaining in our walk, we have now reduced our problem to one of our three previous cases. For each of the cases, we have:

1. If $l < n - i$, then we know we will end our walk on point p_j where $j = l + i < n$. Thus, we have that $i + k = i + l + an = j + an$.

2. If $l = n - i$, then we know we will end our walk on point p_0. Thus, we have that $i + k = i + l + an = i + (n - i) + an = (a+1)n + 0$.

3. Finally, if $n - i < l \leq n$, then we know we will end our walk at point p_j where $j = l - (n - i)$. Thus, we have that $i + k = i + l + an = l + i - n + (a+1)n = l - (n - i) + (a+1)n = j + (a+1)n$.

---

3 Activation Cycle Machines

With a few, minor modifications to cycle graphs, we can create *activation cycle machines*; and in doing so, bring ourselves one step closer to understanding awkward state machines. So what exactly is an activation cycle machine? Let us answer that question by first looking at the formal definition; and then taking some time to examine its meaning afterward.

Definition
An *activation cycle machine* is a state machine with $n > 1$ states, $s_0, s_1, ..., s_{n-1}$, such that for any state, s_k where $k < n-1$, state s_k has only a single transition to state s_{k+1}; and state s_{n-1} has only a single transition to s_0. Furthermore, all activation cycle machines always start at state s_0.

The first a states, where $1 \leq a < n$, of an activation cycle machine are called *activators*. An activation cycle machine is said to be *active* when it's current state is one of it's activators, and is called *inactive* when it's not active. We write $C_{n,a}$ to refer to the activation cycle machine with n states and a activators.

When an activation cycle machine's current state is s_k, for any k, we call k its *position*, and write $\overline{C}(j)$ to refer to the position of the machine after it's j^{th} transition.

The first part of our definition is almost exactly the same as our definition for a cycle graphs. However, instead of talking about points in a graph, and the edges between those points; we define the states of a machine, and the transitions between those states. With such similar definitions, it seems reasonable that we can represent the states of an activation cycle machine by using a cycle graph. In fact, we'll begin our study of activation cycle machines by proving just that.

The definition of an activation cycle machine extends a bit beyond that of cycle graphs by incorporating the notion of *activators*, which is merely a label applied to the first a states of the machine, $s_0, ..., s_{a-1}$. Furthermore, our definition stipulates that while there is always at least one activator, there are never as many activators as there are states in the machine. As such, every activation cycle machine will become *active* at least once, as well as *inactive* at least once as it transitions between its states.

Before we dive into our study of activation cycle machines, below you will find two working examples of an activation cycle machine coded in *ruby*. The first example is more condensed. Rather than keep track of states directly, the first example simulates an activation cycle machine by keeping track of the current position of the machine using arithmetic. Our second example, on the other hand, is represented using states, and as such, needn't rely on arithmetic to function. Whether or not you program, I encourage you to look through both examples carefully; I think you will find they provide a bit of color to our abstract definition.

```ruby
class ActivationCycleMachine
  def initialize(number_of_states, number_of_activators)
    unless number_of_states > 1
      raise 'There must be more than one state'
    end

    unless number_of_activators.positive?
      raise 'The number of activators must be positive'
    end

    unless number_of_activators < number_of_states
      raise 'The number of activators must be less ' \
            'than the number of states'
    end

    @number_of_states = number_of_states
    @number_of_activators = number_of_activators
    @position = 0
  end

  def next_state
    if @position == @number_of_states - 1
      @position = 0
    else
      @position = @position + 1
    end
  end

  def active?
    @position < @number_of_activators
  end

  def inactive?
    !active?
  end

  def position
    @position
  end
end
```

As you can see, our first example simply increments the position of our state machine by one to simulate a transition, with the exception that it resets the position to 0 once its gotten to the maximum position. Since this example does not use states, it does not have a direct understandg of an *activator*. Despite that, the implementation is able to determine activeness indirectly by checking if it's current position is less than the number of activators.

We'll need to break our second example into two seperate classes. Within the first class, we will represent a single state in our machine, which will be implemented as a node in a linked list would. Our second class will be the representation of the activation cycle machine, which we implement much the same way that we would a linked list.

```
class State
  def initialize(is_activator, position)
    @is_activator = is_activator
    @position = position
  end

  def activator?
    @is_activator
  end

  def next_state=(next_state)
   @next_state = next_state
  end

  def next_state
   @next_state
  end

  def position
   @position
  end
end
```

```
class ActivationCycleMachine
  def initialize(number_of_states, number_of_activators)
    unless number_of_states > 1
      raise 'There must be more than one state'
    end

    unless number_of_activators.positive?
      raise 'The number of activators must be positive'
    end

    unless number_of_activators < number_of_states
      raise 'The number of activators must be less ' \
            'than the number of states'
    end

    initial_state = State.new(true, 0)
    current_state = initial_state
    (1...number_of_states).each do |i|
      activator = i < number_of_activators
      current_state.next_state = State.new(activator, i)
      current_state = current_state.next_state
    end
    current_state.next_state = initial_state

    @current_state = initial_state
  end

  def next_state
    @current_state = @current_state.next_state
  end

  def active?
    @current_state.activator?
  end

  def position
    @current_state.position
  end
end
```

As you can see, our second example matches our formal mathematical definition very closely. In particular, the activation cycle machine, once initialized, only has a notion of a current state. The machine is operated by simply replacing the current state with its next state, thus simulating a transition. Furthermore, since the states themselves encode whether or not its an activator; the activation cycle machine is able to determine if its active by simply asking if its current state is an activator; matching our mathematical definition exactly.

Without further ado, let us now tie activation cycle machines and cycle graphs together.

Lemma
Let $C_{n,a}$ be any activation cycle machine.
Let $S = \{ \ s_i \mid 0 \leq i < n \ \}$ be the states of $C_{n,a}$.

Let G_n be the cycle graph of n points.
Let $P = \{ \ p_j \mid 0 \leq j < n \ \}$ be the points of G_n.

Let $\gamma : P \rightarrow S$ be the mapping from the points in our cycle graph to the states in our activation cycle machine such that $\gamma(p_k) = s_k$.

Let $\theta : S \rightarrow P$ be the mapping from the states of our activation cycle machine to the points in our graph such that $\theta(s_k) = p_k$.

Let $\omega : (P, \mathbb{N}) \rightarrow P$ such that $\omega(p_i, k) = p_j$, where p_j is the point arrived to after a walk of length k.

Let $\sigma : (S, \mathbb{N}) \rightarrow S$ such that $\sigma(s_i, k) = s_j$, where j is the position of the machine after k transistions when starting at state s_i.

Then $\gamma(\omega(p_i, 1)) = \sigma(s_i, 1)$.

In other words, for any activation cycle machine, we can determine the next state to be transitioned to by instead taking a walk of length one, starting at the point with the same index as the starting state, around the cycle graph with the same number of points as states in our machine; and then finally mapping the resulting point from our walk back to the states in our machine using γ.

Proof
Our proof is rather trivial and will come directly from our definitions. Let us assume we have some activation cycle machine with n states, $S = \{s_0, s_1, ..., s_{n-1}\}$, and the cycle graph with n points, $P = \{p_0, p_1, ..., p_{n-1}\}$. Let us define the mapping $\theta : S \rightarrow P$ as $\theta(s_k) = p_k$, and it's inverse $\gamma : P \rightarrow S$ which is clearly $\gamma(p_k) = s_k$.

To begin, we will show that if our activation cycle machine is on any state, s_k, that we can determine the next state our machine will transition to by taking a walk of length 1 starting at point $\theta(s_k) = p_k$ and mapping the resulting point, p_j, back to state $\gamma(p_j) = s_j$.

Let our activation cycle machine be at position $n-1$. If we were to take a walk of length 1 from point $\theta(s_{n-1}) = p_{n-1}$, then we will arrive at point p_0 by definition of a cycle graph. Mapping point p_0 back to our cycle machine under γ yields state $\gamma(p_0) = s_0$. By definition of an activation cycle machine, we know that state s_{n-1} transitions to state s_0, thus our mappings hold for $n-1$.

Now let us assume our activation cycle machine is at position $k < n-1$. If we were to take a walk of length 1 from point $\theta(s_k) = p_k$, then we will arrive at point p_{k+1} by definition of a cycle graph. Mapping point p_{k+1} back to our cycle machine under γ yields state $\gamma(p_{k+1}) = s_{k+1}$. By definition of an activation cycle machine, we know that state s_k transitions to state s_{k+1}, thus our mappings hold for all k.

With the proof of this simple lemma, we can immediately apply all our knowledge of cycle graphs to activation cycle machines. Most importantly, we gain the ability to predict the position of any activation cycle machine after any number of transitions.

Corollary
For any n, for any k, the position of the activation cycle machine with n states after k transitions starting from any position j can be determined by taking a walk of length k starting from point p_j around the cycle graph with n points.

Proof
Since we can predict the resulting state of a transition of our cycle machine by instead taking a walk of length 1 on our corresponding cycle graph, it should be obvious that we can also predict the resulting state of our cycle machine after k transitions by instead taking a walk of length k on our cycle graph.

We shall prove our corollary by induction on k, the number of transitions our state machine undergoes. Our base case, $k = 1$, is the direct result of our previos lemma. As such, let us assume for all $j \leq k$, that the resulting position of our cycle machine after j transitions from any position, i, can be determined by instead taking a walk of length j starting at point p_i on our cycle graph and mapping the resulting point back to our cycle machine under our mapping $\gamma : P \to S$, $\gamma(p_l) = s_l$.

To transition $k + 1$ states, let us first transition k states from our position, i. By assumption, we know that our resulting state will be given by taking a walk from point p_i of length k and mapping the resulting point, p_l, back to our machine under γ, giving us state $\gamma(p_l) = s_l$. We now only have one transition left to complete our $k + 1$ transitions. By our lemma, we know this final transition from our state s_l is given by taking extending our walk one more step from p_l and mapping the result back to our machine. As such, mapping the result of a walk of length $k + 1$ around our cycle graph does in fact result in the same position as $k + 1$ transitions would have.

Corollary
For any n, for any k, the position of the activation cycle machine with n states after k transitions starting from any position j is given by $(j + k)\ mod\ n$.

Proof
By our previous corollary, we know the position of our cycle machine after k transitions starting at any position, j, can be determined by instead taking a walk of length k starting from point p_j around the cycle graph with n points.

By our previous theorem on cycle graphs, we know our walk of length k will end at point p_i, where $i = (j + k)\ mod\ n$. As such, the position of our cycle graph after k transitions will also be i.

Lemma
For any activation cycle machine $C_{n,a}$, for any k, the activation cycle machine is active if and only if $a > (k\ mod\ n)$.

Proof
By our previous corollary, we know that the position of our cycle machine, $C_{n,a}$, after k transitions will be given by $j = (k + i)\ mod\ n$, where i is the initial position of our machine. By definition of an activation cycle machine, we know that $i = 0$, thus, our position after the first k transitions is given

by $j = k \ mod \ n$.

Furthermore, an activation cycle machine is defined to be active whenever its current state is one of its activators, in other words, whenever its position is less than a. Putting the two together, we get that our cycle machine is active after the initial k transitions whenever $a > k \ mod \ n$

---

4 Awkward State Machines

Now that we have our definitions for cycle graphs and ACMs, we are almost ready to formally define the *awkward state machine* or *ASM*. An ASM is a state machine for the purpose of generating a set of ACMs using a pre-existing set of ACMs. In order to accomplish this task, with each step of an ASM, all the pre-existing ACMs state's are incremented by one; if none of the ACMs are active after moving, then a new ACM is added to the ASM and it's position is set to 0. Thus, every state of an ASM has at least one ACM that is active.

Every ASM starts with an initial ACM set to position 0, and an *activation branch* of length one greater than the ACM. An activation branch is simply an ACM that has the edge removed from its last point to its inital point; thus an activation branch forms a line instead of a circle. Whenever the ASM must create a new ACM, it does so by creating a copy of its activation branch, then adds the missing edge to the original branch in order to form the new ACM required by the machine. Furthermore, a new node is added to the end of the copied activation branch. If the ASM does not need to create a new ACM after moving, then a new node is added to the end of its activation branch. Thus, the length of an ASM's activation branch increases by one with every iteration.

This section will explore several fundamental questions about ASMs and the sets of ACMs they generate. We will find that the sets of ACMs generated by ASMs have a unique and unexpected relationship to modular arithmetic. For instance, we will see that the most basic ASM generates the prime numbers; and all other ASMs generate sets of integers just as strange, or awkward, as the primes. Furthermore, in our final theorem of this section, we will show that every ASM will generate an infinite set of ACMs if left running.

We will begin this section by formally defining the activation branch and proving that we can indeed create an ACM by adding an edge to it. From there, we will formally define ASMs and then begin our exploration.

Definition
An *activation branch* is directed graph that forms a line. Explicity, the activation branch of length n has points $\{ p_0, p_1 ..., p_{n-1} \}$ such that for every point $p_{i<n-1}$ has a single edge connecting it to p_{i+1}, and p_{n-1} does not have any edges extending from it.

Furthermore, an activation branch is equipped with $a > 0$ activator points: $A = \{ p_i \mid 0 \leq i < a < n \}$.

We denote the activation branch with length n and a activator nodes $B_{n,a}$.

Lemma
For any $B_{n,a}$, adding an edge extending from p_{n-1} to p_0 produces the cycle graph CG_n for ACM $C_{n,a}$.

Proof
By definition of $B_{n,a}$, all points $p_{i<n-1}$ have the same edges as the first $n-1$ points in CG_n for ACM $C_{n,a}$.

Furthermore, the first a points of $B_{n,a}$ are it's activator points, which are the activators of $C_{n,a}$.

Thus, the only edge missing from $B_{n,a}$ that is contained within CG_n is from p_{n-1} to p_0.

Therefore, adding the edge will convert $B_{n,a}$ into CG_n for $C_{n,a}$.
Q.E.D

Definition
An *awkward state machine* (ASM) is state machine running on top of a directed graph composed of a set of ACMs and a single activation branch.

Every ASMs initial state contains a single ACM, $C_{m,a} = C_0$, and the activation branch, $B_{m+1,a}$.
Furthermore, the position of C_0 on the initial state is 0.
We denote the ASM with an initial state containing $C_{m,a}$ as $S_{a,n=m-a}$.

To progress from state $i-1$ to state i for ASM, S, first move every ACM in S to it's next state.
If none of the ACMs in S are active after moving to their next state, then:

1. Create a copy of the activation branch, giving you branches B and B‘.
2. Convert B into an ACM, C, by adding an edge to it's last point, thus leaving a single activation branch B‘. Set the position of C to 0. Now C is contained within the set of ACMs for S.

Regardless of whether one of the ACMs were active, add a new point to the end of the activation branch. Explicitly, if the activation branch had length n, then add an edge extending from p_{n-1} to the new point p_n, thus creating an activation branch of length $n+1$.

If none of the ACMs were active after moving them to their next state, we say that S is *inactive* after moving; otherwise we say S is *active* after moving. Therefore, we only add a new ACM to S when S is inactive after moving.

We denote the inition cycle of an ASM C_0, the first discovered cycle C_1, and the nth discovered cycle C_n.

We denote the graph of ASM $S_{a,n}$ on state k as S^k.

If $C_{p,a}$ is discovered by ASM $S_{a,n}$ on some step k, then we say that $C_{p,a}$ is *discoverable* by S.

Definition
We'll define $[S^a] = \{\ C_i$ within the graph of S on step $a\ \}$,
and $[S] = \{\ C_0$ and all C_i disoverable by $S\ \}$. We refer to $[S]$ as the *school* of cycles for 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 $j - i = 1$.
Thus, $|C_i| \geq |C_j| + a = |C_j| + (j-i)a$.

Assume for k, $i < k \leq j$, 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

Corollary
For any cycle C_j of an ASM, $S_{a,n}$, C_j has at least $n+ja$ non-activator nodes.

Proof
C_0 is defined to have n activator nodes.

Let $C_{j>0} \in [S_{a,n}]$.

Then $|C_j| \geq |C_1| + (j-1)a \geq (|C_0| + a) + (j-1)a$
$= (2a+n) + (j-1)a = n + (j+1)a$.

Thus, after removing the a activator nodes from C_j, there are at least $n+ja$ non-activator nodes left.
Q.E.D

Lemma
Every ASM discovers at least one cycle.

Proof
Let C_0 by the initial cycle for ASM, $S_{a,n}$.

After taking a steps from the initial state of the S, the ASM will be on it's first non-activator node.

Since C_0 is the only cycle, the ASM would be inactive, thus a new cycle would be created.
Q.E.D

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

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:

$$\overline{C^k_0} = k\ mod\ |C_0| = [k + (a + n)]\ mod\ (a + n) = (k + a + n)\ mod\ |C_0|.$$

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^k_i is given by:

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

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 $C_i \in [S^k_{a,n}]$ is given by $\overline{C^k_i} = (k + a + n)\ mod\ |C_i|$ for any C_i in $[S^k]$.
Q.E.D

Corollary

For any cycle $C_j \in [S_{a,n}]$, moving $k|C_j|$ steps will maintain C_j's position.

Proof
Let $C_j \in [S]$.
Assume S is on step p.

Then $\overline{C_j} = p + a + n \ mod \ |C_j|$.
Thus, if we move $k|C_j|$ steps from p, the position of C_j will be given by:

$p + a + n + k|C_j| \ mod \ |C_j| = p + a + n \ mod \ |C_j|$.

Therefor C_j's position was maintained.
Q.E.D

Lemma
For any cycle $C \in [S]$, if $k \geq |C|$, then the position of C after moving k steps from C's discovery is the same as moving $k \ mod \ |C|$ steps.

Proof
Assume $C \in [S]$, and S is on step p.
Then the position of C is given by $p + a + n \ mod \ |C|$.

Let $k > |C|$, $b = k \ mod \ |C|$.

The position of C on step $p + k$ is given by:
$p+k+a+n \ mod \ C = p+b+a+n \ mod \ C$ which is the position after moving b steps from p.
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
For any $C_i \in [S_{a,n}]$, $|C_i|$ is the least positive integer greater than $|C_{i-1}|$ such that $|C_i|\ mod\ |C_j| \geq a$ for all $j < i$.

Proof
Assume you discovered C_{i-1} on step p.
Assume k is the first step after discovering C_{i-1} such that all cycles are inactive (if we can show that cycle C_i is discovered on step k, then our proof will be complete).

Then, on all steps, q, $p \leq q < k$, there must exist at least one cycle C_j that is active (implying $q + a + n\ mod\ |C_j| < a$). Therefore, the ASM algoirthm cannot create cycle C_i on any step $p \leq q < k$.

Furthermore, on step k, the algorithm must create C_i, giving us $\overline{C_i^k} = 0 = k + a + n\ mod\ |C_i|$.

Thus, since $k + a + n = |C_i|$ is the least positive integer such that $|C_i|\ mod\ |C_j| \geq a$ for all $j < i$, and have proved our lemma.
Q.E.D

Definition
A positive integer is prime if it's only factors are 1 and itself.

Axiom
The ith prime number, p_i, is the least integer that is greater than the $(i-1)th$ prime number and is not divisible by p_j for all $j < i$ (with the first prime number $p_0 = 2$).
Note: This is only an axiom because proving the statement is distracting to ASMs.

Lemma
$|S_{1,1}| = P = \{$ the set of prime numbers $\}$

Proof
$S_{1,1}$ starts with an initial cycle of length $1 + 1 = 2$, thus giving us the first prime number.

For all i, j such that $i > j \geq 0$, C_i will have the property that $|C_i|$ *mod* $|C_j| \geq 1$, equivalenty, $|C_i|$ is not divisible by any of the previous lengths $|C_j|$. Furthermore, $|C_i|$ is the smallest such integer greater than $|C_{i-1}|$ with the property of not being divisible by the previous cycle lengths.

Thus, by our axiom, $S_{1,1}$ produces the prime numbers.
Q.E.D

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.
Let s be the number of cycles in $[S_{a,n}]$.

Then there exists some step p on which the last cycle, $C_{max} = C_{s-1}$ of the ASM is discovered.

We know C_{max} must have at least $n+(s-1)a > (s-1)a$ non-activator nodes.

We also know that on step p, all cycles $C_{i<s}$ must have been inactive to have discovered C_{max}.

Furthermore, on step j, C_{max}'s position is at 0.

Let $z = \prod_{i<s} |C_i|$.

If we move the ASM z steps, then the position of every $C_{i<z}$ will be maintained since z is a multiple of the length of every $C_{i<z}$.

Thus, if after moving z steps from p, C_{max} is inactive, then we would need to create a new cycle, and we'd be finished with our proof.

Assume C_{max} was active after moving the z steps.

Let $n = \overline{C_{max}}$

We know $n < a$ since C_{max} is inactive.

Note: the position of C_{max} after moving zt steps is the same as after moving nt steps since $z > |C_{max}|$ (if z were less than $|C_{max}|$, then moving z steps would have inactivated C_{max} since $z > a$).

Let j be the greatest integer such that $jn < a$.
Then your position after moving jz steps will be $a - jn$.
Thus, the $(j + 1)z$th step will put C_{max} at position $(a - jn) + n \geq a$.

If the ASM is to remain active, then $(a-jn)+n$ must go beyond all $(s-1)a$ non-activator nodes in C_{max}.

Thus, $(a - jn) + n > a + (s - 1)a \geq sa \geq 2a$.

But both n and $(a - jn)$ are less then a.
But that means: $a + a = 2a > (a - jn) + n \geq 2a$, which is a contradiction!

Therefore, moving $(j+1)z$ steps from the discovery of C_{max} will maintain the (inactive) positions of all $C_{i<z}$ and also move C_{max} into one of it's inactive nodes, requiring a new cycle to be made, and completing our proof.
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