Awkward State Machines

Will Dengler

February 7, 2021

1 Introduction

TODO motivate the reader

2 Notations & Assumed Knowledge

The study of awkward state machines requires that we make use of preexisting theory. In this section, we will specify any such knowledge that is required.

2.1 Notations

Notation: Sets

- $\mathbb{Z}$ to represent the set of integers.
- $\mathbb{N}$ to represent the set of natural numbers, which we will define to be the non-negative integers.
- $\mathbb{N}^+$ to represent the set of positive nature numbers
- $[n]$ to represent all natural numbers less than n.
- $x \in Y$ to mean x is an element of the set Y.

2.2 The Remainder Theorem

We will need to borrow the concept of the *remainder* from basic number theory. We will not bother proving the associated theorem ourselves, as it

is well known, see https://en.wikipedia.org/wiki/Remainder for a basic overview.

The Remainder Theorem
For any positive integer $n \in \mathbb{N}^+$, for any non-negative integer $t \in \mathbb{N}$, there exists a unique integer $q \in \mathbb{N}$, and a unique integer $r \in [n]$ such that $t = qn + r$.

We call r the *remainder* of t when divided by n.

It should be noted that there is a wealth more knowledge known about the remainder; however, we will take the time to prove any other results we might need throughout the rest of this paper.

2.3 Graphs

https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)
https://en.wikipedia.org/wiki/Directed_graph

3 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.

3.1 Basic Definitions

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

$$P = \{ p_i \mid i \in [n] \}$$

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.

3.1.1 The Walk Functions

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.

We define $\omega(p_i, 0) = p_i$ for completeness and consistency, even though a walk of length 0 does not exist. We can instead interpret this as the result of not taking a walk.

Furthermore, we define the *walk index function*, $\Omega : (P, \mathbb{N}) \to [n]$ 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 definition 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:

Lemma
For any cycle graph G_n:

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

3.2 Validity of the Walk Function

In order for our mappings ω and Ω to be valid mappings, they will have defined for all input; furthermore, each input can only map to a single output. Since our functions map to the outcomes of walks of varying length around cycle graphs, they will only be valid if for any walk length, starting at any point, on any cycle graph, that it must be the case that the walk can be completed; furthermore, there must only be a single walk that can be taken. 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 other words, 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 demands us to show that there is only a single walk of length one starting at any point, in any cycle graph. But of course this is true because our definition of a cycle graph stipulates that the point p_{n-1} has only a single edge going to point p_0, while all other points p_k only have a single edge going to point p_{k+1}. As such, a walk of length one is possible from any point; furthermore, there is only one such walk since there is only a single edge that can be traversed.

Inductive Hypothesis
Let us assume that for all $1 \leq j < k$, for all cycle graphs, that it is the case that there is only a single walk that can be taken of length j starting from any point within the graph. In other words, that $\omega(p_i, j)$ is defined on all points of all cycle graphs whenever $j < k$.

Inductive Step
We must now show that we can complete a walk of length k starting from any point in any cycle graph, and that there is only one walk that we can take to do so.

Our inductive hypothesis tells us that there is only a single walk of length $k-1$ we can take starting from any point p_i, ending at point $\omega(p_i, k-1)$.

Furthermore, our base case tells us that there is only a single walk of length one that can be taken from any point in our graph. As such, it must be the case that there is only a single walk of length one that can be taken from point $\omega(p_i, k-1)$; equivalently we know $\omega(\omega(p_i, k-1), 1)$ is defined.

Since our composite walk $\omega(\omega(p_i, k-1), 1)$ is composed of a walk of length $k-1$ and a second walk of length 1, our composite walk has length k. Furthermore, since both components of our walk were the only walks that could have been taken, we can conclude that our composite walk is the only walk that could have been taken of length k. As such, not only have we completed our proof, but we have also shown that:

$$\omega(p_i, k) = \omega(\omega(p_i,\ k-1),\ 1) \text{ whenever } k > 1$$

3.3 Composition of Walks

In order to show a walk of a certain length has some property, we will often need to "split up" the walk into two or more shorter walks and then reason about the composition of the shorter walks in order to reason about the original longer walk.

To start, we will show that a walk of any length j is equivalent to first taking a walk of length $t < j$, followed by a walk of length $j-t$; or first taking a walk of length $j-t$, followed by a walk of length t.

Lemma
For any cycle graph G_n, for any point p_i, for any integers $0 \leq t \leq j$,:

$$\omega(p_i, j) = \omega(\omega(p_i, t), j-t) = \omega(\omega(p_i, j-t), t).$$

Proof
We know there exists a single walk of length t starting at point p_i. Let p_w be the point this walk ends at, then $p_w = \omega(p_i, t)$.

We know there exists a single walk of length $j-t$ starting from point p_w. Let p_y be the point this walk ends at, then $p_y = \omega(p_w, j-t)$.
Applying substitution yields:

$$p_y = \omega(\omega(p_i, t), j - t)$$

Combining the two above walks yields a walk from p_i to p_y of length $t + (j - t) = j$. Thus, we have shown:

$$p_y = \omega(p_i, j) = \omega(\omega(p_i, t), j - t)$$

To complete our proof, we need to show:

$$\omega(p_i, j) = \omega(\omega(p_i, j - t), t).$$

Let us define $a = j - t$.

From the first part of this proof, we know that

$$\omega(p_i, j) = \omega(\omega(p_i, a), j - a)$$

Substituting out a yields:

$$\omega(p_i, j) = \omega(\omega(p_i, j - t), j - (j - t)) = \omega(\omega(p_i, j - t), t).$$

Corollary
For any cycle graph G_n, for any point p_i, for any non-negative integers t, v:

$$\omega(p_i, t + v) = \omega(\omega(p_i, t), v) = \omega(p_i, v), t)$$

Proof
Let $j = t + v$. Through the previous lemma, we know that

$$\omega(p_i, t + v) = \omega(p_i, j) = \omega(\omega(p_i, t), j - t)$$
$$\omega(p_i, t + v) = \omega(p_i, j) = \omega(\omega(p_i, j - t), t)$$

Substitution brings us:

$$\omega(p_i, t + v) = \omega(\omega(p_i, t), (t + v) - t) = \omega(\omega(p_i, t), v)$$
$$\omega(p_i, t + v) = \omega(\omega(p_i, (t + v) - t), t) = \omega(\omega(p_i, v), t)$$

3.4 Shortest Walks

Lemma
For any cycle graph G_n, for any integers i, j such that

$$0 \leq j \leq i < n$$

then

$$\omega(p_j, i - j) = p_i$$

Proof
This will be a proof by induction on the length of the walk $l = i - j$.

Base Case ($length = 0$)
Let $l = 0$ be the length of the walk, giving us $l = i - j = 0$ and $i = j$.
Substituting gives us:

$$\omega(p_j, l) = \omega(p_j, i - j) = \omega(p_j, 0) = p_j = p_i$$

Hypothesis
Assume for all $l < t \leq n - l$, that $\omega(p_j, l) = \omega(p_j, i - j) = p_i$

Inductive Step
We need to show that $\omega(p_j, t) = \omega(p_j, i - j) = p_i$.

By a previous lemma, we know that:

$$\omega(p_j, t) = \omega(\omega(p_j, t - 1), 1)$$

Focusing on the inner function and substituting:

$$\omega(p_j, t - 1) = \omega(p_j, (i - j) - 1) = \omega(p_j, (i - 1) - j)$$

Applying the inductive hypothesis:

$$\omega(p_j, t - 1) = \omega(p_j, (i - 1) - j) = p_{i-1}$$

Substituting p_{i-1} back into $\omega(p_j, t)$ yields:

$$\omega(p_j, t) = \omega(\omega(p_j, t - 1), 1) = \omega(p_{i-1}, 1) = p_i$$

Corollary
For any cycle graph G_n, for any integers i, j such that

$$0 \leq j < i < n$$

the shortest walk from p_j to p_i has length $i - j$.

Proof
Our lemma proved that $\omega(p_j, i - j) = p_i$, for $0 \leq j \leq i < n$. As such, we need to show that for any $l < i - j$, that $\omega(p_j, l) \neq p_i$.

Let $1 \leq l < i - j$. Let x be the integer such that $l = (i - j) - x$. Then,

$$\omega(p_j, l) = \omega(p_j, (i - j) - x) = \omega(p_j, (i - x) - j)$$

Our lemma tells us that whenever $j < i - x$, then $\omega(p_j, (i - x) - j) = p_{i-x}$. From $l = (i - j) - x$, we get $x = (i - j) - l$. Substituting,

$$i - x = i - ((i - j) - l) = i - i + j + l = j + l$$

By stipulation, we have $1 \leq l$. Therefore, $j < j + 1 \leq j + l = i - x$.

With our inequality shown, we can state $\omega(p_j, l) = p_{i-x}$ for all possible l.

If we can show it is always the case that $x > 0$, then $p_{i-x} \neq p_i$, and we will have completed our proof.

By stipulation, we know that

$$l = (i - j) - x < i - j$$

We can subtract $(i - j)$ to get $-x < 0$.

Finally, we can multiply by -1, yielding $x > 0$.

Corollary
For any cycle graph G_n, for any non-negative integers i, j such that $i + j < n$,

$$\Omega(p_i, j) = i + j$$

Proof
Since we have that $i \leq i + j < n$, the previous lemma tells us:

$$\omega(p_i, (i+j) - i) = p_{i+j}$$

We can simplify $(i+j) - i = j$, thus we have:

$$\omega(p_i, j) = p_{i+j} \text{ and } \Omega(p_i, j) = i + j.$$

Lemma
For any cycle graph G_n, for any integers i, j such that

$$0 \leq i \leq j < n - 1$$

then

$$\omega(p_j, (n-j) + i) = p_i$$

Proof
We shall prove this directly. Using the previous lemma, we know that:

$$\omega(p_j, (n-1) - j) = p_{n-1} \text{ since } j \leq (n-1)$$

Furthermore, we know that:

$$\omega(p_{n-1}, 1) = p_0 \text{ by structure of } G_n$$
$$\omega(p_0, i) = \omega(p_0, i - 0) = p_i \text{ since } 0 \leq i$$

As such, we can create a composite of these three walks to form a walk from p_j to p_i having length:

$$((n-1) - j) + 1 + i = (n-j) + i$$

which completes our proof.

Corollary
For any cycle graph G_n, for any point p_i, for any non-negative integer $j \leq n$ such that $i + j \geq n$, then

$$\Omega(p_i, j) = j - (n - i) = (i + j) - n$$

Proof
If we can show that $0 \leq j - (n-i) \leq i$, then the previous lemma tells us that:

$$\Omega(p_i, (n-i) + (j - (n-i))) = j - (n-i)$$

Simplifying our expression yields:

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

Thus, if we can show $0 \leq j - (n-i) \leq i$, then we will have:

$$\Omega(p_i, (n-i) + (j - (n-i))) = \Omega(p_i, j) = j - (n-i)$$

We can subtract i from our inequality, $i + j \geq n$, to get $j \geq n - i$. As such, it is the case that

$$j - (n-i) \geq (n-i) - (n-i) \geq 0$$

Now we just need to show that $j - (n-i) \leq i$ to complete our proof. The truth of this follows from the fact that $j \leq n$:

$$j - (n-i) = (j+i) - n \leq (n+i) - n = i$$

Corollary
For any cycle graph G_n, for any integers i, j such that

$$0 \leq i \leq j \leq n-1$$

the shortest walk from p_j to p_i has length $(n-j) + i$.

Proof
We will complete this proof by showing that $\omega(p_j, l) \neq p_i$ for all $l < (n-j)+i$.

We will start by assuming we have $l \leq (n-j)$. For any such l, we can find some k such that $l = (n-j) - k$ where $0 \leq k < n - j$. Substituting yields:

$$\omega(p_j, l) = \omega(p_j, (n-j) - k) = \omega(p_j, (n-k) - j)$$

By a previous lemma, we know that $\omega(p_j, (n-k) - j) = p_{n-k}$ as long as $n - k > j$.

We know that $k < n - j$ by our selection of l. Adding j yields $k + j < n$, followed by subtracting k give us $j < n - k$. Thus, we are able to apply our lemma to give us $\omega(p_j, (n-k) - j) = p_{n-k}$. Furthermore, we know $i \leq j < n - k$, thus we have that $i \neq n - k$, yielding:

$$\omega(p_j, l) = \omega(p_j, (n-k) - j) = p_{n-k} \neq p_i.$$

As such, we have eliminated all $l \leq (n - j)$.

We now need to show that if l is within: $n - i < l < (n - 1) + i$, then $\omega(p_j, l) \neq p_i$.

For any l, we can find some k such that $l = (n - j) + k$ where $1 \leq k < i$. Substituting allows us to apply the previous lemma:

$$\omega(p_j, l) = \omega(p_j, (n - j) + k) = p_k \neq p_i.$$

Corollary
For any cycle graph G_n, for any points p_i, p_j, there exists positive integers $a, b \leq n$ such that:

$$\omega(p_i, a) = p_j \text{ and } \omega(p_j, b) = p_i.$$

In other words, it is always possible to walk from any point p_i to any point p_j with a length no greater than n.

Furthermore, whenever $p_j \neq p_i$, then the integers a, b will satisfy the relationship: $a = n - b$ (equivalently $b = n - a$).

Finally, whenever $p_j = p_i$, then we will have $a = b = n$.

Proof
The first part of this proof follows directly from the previous two lemmas.

Let p_i, p_j be any two points such that $i < j$. We know that $\omega(p_i, j - i) = p_j$ by the first of our two previous lemmas. Furthermore, $j - i < j < n$.

Now assume that $j \leq i$. We know that $\omega(p_i, (n - i) + j) = p_j$ by the second of our two previous lemmas. Furthermore, $(n - i) + j \leq (n - i) + i = n$.

As such, we completed showing that from any point p_i, there exists a walk to any other point p_j with a length less than or equal to n.

We will now prove that whenever $p_j \neq p_i$, that $b = n - a$.

Assume that $i < j$. We know that $\omega(p_i, j-i) = p_j$, and $\omega(p_j, (n-j)+i) = p_i$. Therefor, it must be the case that

$$a = j - i$$
$$b = (n - j) + i = n - (j - i) = n - a$$

We will conclude this proof by showing that a walk of length n from any point p_i will return to point p_i.

By the previous lemma, we know a walk of length $(n - i) + i) = n$ will end on point p_i since $i \leq i$.

3.5 Repetition of Walks

Lemma
For any cycle graph G_n, for any point p_i, for any integers $t \geq 0, a \geq 1$

$$\omega(p_i, t) = \omega(p_i, t + an)$$

Proof
We shall complete this proof by induction on a.

Base Case $(a = 1)$
Let p_i be any point in our graph, let $t \geq 0$, and let $p_k = \omega(p_i, t)$.

By the previous corollary, we know that $\omega(p_k, n) = p_k$. As such, the composition of these walks is $\omega(\omega(p_i, t), n) = \omega(p_i, t + n) = p_k = \omega(p_i, t)$, thus we have completed our base case.

Inductive Hypothesis
Assume for all $1 \leq a < l$ for some l, that $\omega(p_i, t) = \omega(p_i, t + an)$.

Inductive Step
We need to show that $\omega(p_i, t + ln) = \omega(p_i, t)$.

By previous lemma, we know:

$$\omega(p_i, t + ln) = \omega(\omega(p_i, n), t + (l - 1)n)$$

By the previous corollary, we know $\omega(p_i, n) = p_i$. As such, we can substitute:

$$\omega(\omega(p_i, n), t + (l-1)n) = \omega(p_i, t + (l-1)n)$$

Since $l - 1 < l$, we can apply our inductive hypothesis to complete our proof

$$\omega(p_i, t + (l-1)n = \omega(p_i, t)$$

3.6 Remainders

We will often wish to refer to remainder throughout our study of awkward state machines. However, it would be rather clumsy to have to constantly to write out that "r is the remainder of t when divided n" to make reference to our definition. As such, let us define a new function, ρ, that will allow us to reference the remainder without all the fuss.

Definition
For any positive integer n, for any integer $t \geq 0$, the *remainder function*, $\rho : (\mathbb{N}^+, \mathbb{N}) \to [n]$, maps (n, t) to the remainder of t when divided by n.

As such, if r is the remainder of t when divided by n, then we write:

$$\rho(n, t) = r$$

Let us quickly note that combining our imported theorem with the remainder function yields: for any positive integer n, and any integer $t \geq 0$, there exists a unique integer $q \geq 0$ such that $t = qn + \rho(n, t)$.

Lemma
For any $i \in [n]$, if $i = j - qn$ for some integers $j, q \in \mathbb{N}$, then $\rho(n, j) = i$.

Proof
Assume $i \in [n]$ and $i = j - qn$ for some integers $j, q \in \mathbb{N}$.

Solving for j yields. $j = i + qn$.

Since $i < n$, then i and q must be the unique integers from the remainder theorem. As such i is defined to be remainder of j when divided by n.

3.7 Omega Walk Theorem

Lemma
For any cycle graph G_n, for any point p_i, for any integer $t \geq 0$

$$\omega(p_i, t) = \omega(p_i, \rho(n, t))$$

Proof
We know there exists some integer $q \geq 0$ such that $t = qn + \rho(n, t)$. As such, we can use this expression to substitute for t:

$$\omega(p_i, t) = \omega(p_i, qn + \rho(n, t)) = \omega(\omega(p_i, qn), \rho(n, t))$$

By previous lemma, we know that:

$$\omega(p_i, qn) = \omega(p_i, qn + 0) = \omega(p_i, 0) = p_i$$

As such, we can substitute in p_i to complete our proof:

$$\omega(p_i, t) = \omega(\omega(p_i, qn), \rho(n, t)) = \omega(p_i, \rho(n, t))$$

We have now accumulated enough knowledge about cycle graphs with a theorem.

Omega Walk Theorem
For any cycle graph G_n, for any point p_i, for any $t \in \mathbb{N}$

$$\Omega(p_i, t) = \rho(n, i + t)$$

In other words, if a walk of length t starting at point p_i ends on point p_j, then $j = \rho(n, i + t)$.

Proof
Let p_i be any point. Let $t \in \mathbb{N}$.

Let $q \in N$ be the integer such that $t = \rho(n, t) + qn$. Then $\rho(n, t) = t - qn$.

The previous lemma tells us that $\Omega(p_i, t) = \Omega(p_i, \rho(n, t))$.

Since $\rho(n, t) < n$, our previous corollaries tell us that

$$\Omega(p_i, \rho(n, t)) = i + \rho(n, t) \text{ whenever } i + \rho(n, t) < n$$
$$\Omega(p_i, \rho(n, t)) = \rho(n, t) - (n - i) \text{ whenever } i + \rho(n, t) \geq n$$

As such, we need to show that both of these cases are in fact equal to $\rho(i+t)$.

Case 1
Let $\rho(n, t) + i < n$, then

$$\Omega(p_i, \rho(n, t)) = i + \rho(n, t) = i + (t - qn) = (i + t) - qn$$

Since $\Omega(p_i, \rho(n, t)) \in [n]$, then $\Omega(p_i, \rho(n, t)) = \rho(i + t)$ by previous lemma.

Case 2
Let $\rho(n, t) + i \geq n$, then

$$\Omega(p_i, \rho(n, t)) = \rho(n, t) - (n - i) = (t - qn) - (n - i) = (i + t) - (q + 1)n$$

Since $\Omega(p_i, \rho(n, t)) \in [n]$, then $\Omega(p_i, \rho(n, t)) = \rho(i + t)$ by previous lemma.

---

4 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.

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

Let $\sigma : (S, \mathbb{N}) \to S$ 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 $\theta : S \to P$ as $\theta(s_k) = p_k$, and it's inverse $\gamma : P \to S$ 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 $a > k \ mod \ n$

---

5 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