Languages, Automata, and Emergent Structure

Oliver Turner

January 1, 1980

Contents

1 An Abstraction for Languages

Mathematics could be described as the study of emergent patterns in abstract structures. As lacking and futile as some might find any attempt at defining something such as Mathematics, this definition does capture some of the activities mathematicians participate in. We often start with a relatively small and simple set of assumptions and then study what they imply and what they allow. We have create sets, vector spaces, categories, groups, and hundreds more abstractions. We have created layers of abstraction by embedding one abstraction into another, what a category theorist might describe as category isomorphisms. Removing unimportant details allows us to generalise, letting us prove facts about *all* functions without having to look at specific ones. Proofs taught in a basic physics course about vectors in space suddenly apply to infinite vector spaces of polynomials. In the following we will look at one such abstraction, *languages*, with particular focus on the class of *automata regular languages*.

What is a language*?* The first line from the Wikipedia article describes them as "a structured system of communication"[1]. How could we capture this in a mathematical abstraction? It helps to look at some examples of some languages we would like to include in our description:

- This is a sentence in English.
- 
- ... — – . / – — .-. ... .

We would also like to include Japanese, Arabic, Nautical flag signaling, programming languages. Even things like postcodes, phone numbers and email addresses are a "structured system of communication", albeit they communicate a very specific set of information, so they should be included.

What do all of these have in common? They are each a sequence of symbols. These symbols could be visual and colourful like the nautical flags. They could be auditory like Morse code. Yet they are symbols nonetheless. This is the first abstraction we will define, the set of symbols that a languages is defined over and with. It doesn't matter what these symbols are, just that we have a finite set of them to pick from. What is this set of symbols but an alphabet?

[1] Language - Wikipedia - https://en.wikipedia.org/wiki/Language

Definition 1.1. An **alphabet** $\mathcal{A}$, is a finite non-empty set containing letters or symbols.

Remark 1.1. Notice that we exclude empty alphabets. What is a language defined with no letters?

Concatenating letters, placing them next to each other in a certain order, gives us words.

Definition 1.2. A **word** w, is a concatenations of letters from an alphabet $\mathcal{A}$, representable by a tuple of letters.

$$w := \begin{cases} \varepsilon, & \text{if } |w| = 0, \\ (w_1, w_2, \ldots, w_n), w_i \in \mathcal{A}, \forall i, & \text{if } |w| > 0, \end{cases}$$

Where $|w|$ is the length of the word and ε is the empty word.

This already links intuitively with mathematical concepts that we are familiar with such as the Cartesian product: a word w is a tuple of letters from $\mathcal{A}$ of length n, so $w \in \mathcal{A}^n$.

A useful concept to define now is the set of all words possible with a certain alphabet. The set containing all words of all lengths possible with the letters from $\mathcal{A}$:

Definition 1.3. The set of all words using an alphabet $\mathcal{A}$ is noted and defined as $\mathcal{A}^* := \bigcup_{n \in \mathbb{N}} \mathcal{A}^n$.

Languages aren't just defined by their alphabet, they have other rules that validate that a word is actually part of that language. For example, although "pahsdlive" is a word made with the English alphabet it is not part of the English language. These rules can be as nebulous and flexible as those of English, or as precise as those of a programming language. How do we capture an infinitely flexible set of arbitrarily complex rules? Do we need to? What if we instead focus on what these rules do and what they create. All languages are a new, possibly infinite, set of words made up of this alphabet defined by their rules.

Definition 1.4. A **language** on $\mathcal{A}$ is any subset $\mathcal{L} \subseteq \mathcal{A}^*$.

2 Finite Automata

Languages are complex. It is implausible, and in some senses uninteresting, to analyse all of them at once. The same way we don't analyse all possible functions simultaneously and instead only look at, for example, all continuous functions, we need to impose some restrictions on our languages. Some rules and properties that we can work with to extract patterns. For this purpose let us look at a different concept: *finite automata*.

Our goal is to create a class of languages that can be defined using these finite automata, so what must our automata be capable of? Firstly, they must be able to process our words, they need to accept any character we provide them that is part of our alphabet. Secondly, they must give us a response, some statement of either acceptance into the language or rejection from the language.

Finite automata are also known as finite state machines, this describes slightly more accurately what they consist of: a finite number of internal states, and some way of changing that state through processing an input. We can combine these requirements and properties into the following construct:

Definition 2.1. A **finite state automaton** $A := (S, S', \mathcal{I}, T)$ has four parts:

- A finite non-empty set of internal **states** S;
- A subset of states $S' \subseteq S$ designated as **accepting states**;
- A finite non-empty set $\mathcal{I}$, called the **input alphabet**, of input symbols;
- A **transition** or **next state** function $T : S \times \mathcal{I} \to S$ that, for each 'current state' $s \in S$ and 'current input' $i \in \mathcal{I}$ returns a 'next state' $T(s, i) \in S$.

Example 2.1. Let us construct a simple automaton $A = (S, S', \mathcal{I}, T)$ with:

- Two states, numbered consecutively, $S = \{1, 2\}$;
- One accepting state, $S' = \{2\}$;
- The *binary alphabet* as input alphabet, $\mathcal{I} = \{0, 1\}$

- A transition function that goes to the state represented by the next input such as the following:

$$\begin{aligned} T : S \times \mathcal{I} &\to S \\ (1,0) &\mapsto 1 \\ (2,0) &\mapsto 1 \\ (1,1) &\mapsto 2 \\ (2,1) &\mapsto 2 \end{aligned}$$

We can represent this automata using a **transition table**. Placing the states along the left and the input alphabet along the top. The values in the table are the state that that row's state transitions to given that column's input. We mark accepting states with a double circle.

		0	1
	1	1	2
◎	2	1	2

Table 1: An example transition table.

This displays the full automata and all of its behaviour, this has similarities to the adjacency matrix for a graph. Like an adjacency matrix we can translate this into a graph to visually represent this table. Figure 1 is the transition diagram of the automata described in Table 1.

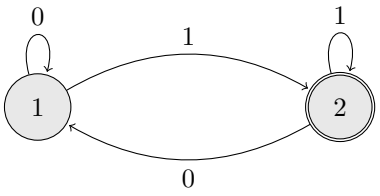


Figure 1: An example transition diagram.

How does this process a word? Given a starting state $s_0 \in S$ and a word $w = (w_1, \ldots, w_n) \in \mathcal{I}^*$ we can process the word one letter at a time. The state after processing the next letter is always the result of the transition function, given our current state and the next letter. So the state after processing the first letter is $s_1(w) = T(s_0, w_1)$ and our state after processing all letters up to and including the k^{th} letter is $s_k(w) = T(s_{k-1}, w_k)$. This lets us define a function that tells us what state our automaton is after processing a given word.

Definition 2.2. Let $A = (S, S', \mathcal{I}, T)$ be a finite state automaton and let $w \in \mathcal{I}^*$. For any state $s \in S$, we define the **final state function**:

$$\begin{aligned} F : S \times \mathcal{I}^* &\to S \\ (s, w) &\mapsto s_{|w|}(w) \end{aligned}$$

So $F(s, w)$ is the final state of A, started from the initial state s, after processing w.

This now lets us define the language described by our automaton given a specific starting state:

Definition 2.3. Let $A = (S, S', \mathcal{I}, T)$ be a finite state automaton with initial state s_0. A word $w \in \mathcal{I}^*$ is **accepted** by (A, s_0) if $F(s_0, w) \in S'$. The **language accepted** by (A, s_0), denoted by $\mathcal{L}(A, s_0)$, is the set of all words that are accepted by A starting from state s_0.

This lets us, given an automaton, define a language, which in turn lets us describe a class of languages:

Definition 2.4. Given a language $\mathcal{L}$, on an alphabet $\mathcal{A}$. Then $\mathcal{L}$ is **automaton regular** if there exists a finite state automaton, A, which has the input alphabet $\mathcal{A}$ and starting state s_0 such that $\mathcal{L}$ is the language accepted by A.

3 Operations on Languages

Any language $\mathcal{L}$ on an alphabet $\mathcal{A}$, as defined in Definition 1.4, is a set in the universe $\mathcal{A}^*$. This means that standard set operations can be used on languages to define new languages: $\mathcal{L}_1'$, $\mathcal{L}_1 \cap \mathcal{L}_2$, $\mathcal{L}_1 \cup \mathcal{L}_2$, $\mathcal{L}_1 \triangle \mathcal{L}_2$. *How do these operations affect the* automaton regularity *of languages?*

Lemma 3.1. *If the language $\mathcal{L}$ is* automaton regular *then so is $\mathcal{L}'$, its* complementary language.

Proof. $\mathcal{L}$ being *automaton regular* means, from Definition 2.4, that there exists a finite state automaton $A_1 = (S, S', \mathcal{I}, T)$ and a starting state s_1 for which $\mathcal{L}$ is the *accepted language*. In other words, the final state of the automaton after "processing" any word is accepting if and only if that word is in the language $\mathcal{L}$.

The *complimentary language* of $\mathcal{L}$, $\mathcal{L}'$, is made up of every word that is not in $\mathcal{L}$. To prove *automata regularity* of $\mathcal{L}'$ we need to construct an automata whose accepting language is $\mathcal{L}'$.

Consider the automata $A_2 = (S_1, S_1 \setminus S_1', \mathcal{I}, T_1)$. As we know, all words in $\mathcal{L}$ end at a state in S_1' and all words not in $\mathcal{L}$ end at a state not in S_1'. That is, they end at a state in $S_1 \setminus S_1'$, which is the accepting set of A_2. This satisfies our condition.

The result of this process is demonstrated on an example automata in Figure 2.

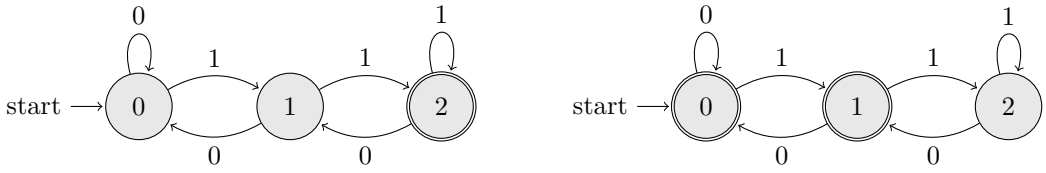


Figure 2: The right automata is the result of inverting the accepting set of the left automata.

□

Lemma 3.2. *If the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ are automaton regular over the same alphabet, then the language $\mathcal{L}_1 \cap \mathcal{L}_2$, that accepts all of the words accepted by both languages, is also* automaton regular.

Proof. The automata constructed for $\mathcal{L}_1 \cap \mathcal{L}_2$ is given below:

$$A_3 = (S_1 \times S_2, S_1' \times S_2', \mathcal{I}, T_3)$$

Where T_3 is the transition function:

$$\begin{aligned} T_3 : (S_1 \times S_2) \times i &\to S_1 \times S_2 \\ (s_1, s_2, i) &\mapsto (T_1(s_1, i), T_2(s_2, i)) \end{aligned}$$

Verifying that this satisfies Lemma 3.2 is left out for the sake of brevity. □

Theorem 3.1. *Using the elementary set operations on automata regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$ result in other automata regular languages.*

The elementary set operations are:

$$\mathcal{L}_1',\ \mathcal{L}_1 \cap \mathcal{L}_2,\ \mathcal{L}_1 \cup \mathcal{L}_2,\ \mathcal{L}_1 \triangle \mathcal{L}_2$$

Proof. The proofs for $\mathcal{L}_1'$ and $\mathcal{L}_1 \cap \mathcal{L}_2$ are Lemma 3.1 and Lemma 3.2 respectively. The other elementary set operations can be constructed using these two[2].

$$\begin{aligned} \mathcal{L}_1 \cup \mathcal{L}_2 &\equiv (\mathcal{L}_1' \cap \mathcal{L}_2')' \\ \mathcal{L}_1 \triangle \mathcal{L}_2 &\equiv (\mathcal{L}_1 \cap \mathcal{L}_2') \cup (\mathcal{L}_1' \cap \mathcal{L}_2) \end{aligned}$$

As all operations used on the right are *automata regularity* preserving, so are the operators on the left. □

[2]They form what is known in Boolean logic as a "universal gate"

4 The Structure of Languages

How does one analyse languages of potentially infinite size? One step at a time. Instead of looking at all words in a language we look at all words of a certain length. This makes each step a finite analysis as the number of words of length n is bounded from above by $|A|^n$ (shown in Lemma 4.1). Firstly, let us define a representation for the number of words of a certain length in the language.

Definition 4.1. The number of words of length n in the language $\mathcal{L}$ is represented by $c_{\mathcal{L}}(n) := |\{w \in \mathcal{L} : |w| = n\}|$

Definition 4.2. The **profile** of a language is the collection of numbers $c_{\mathcal{L}}(n)$ for $n = 0, 1, 2, \ldots$.

Example 4.1. Given the language of even base ten numbers. (So using an alphabet of $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$).

$$\begin{aligned} c_{\mathcal{L}}(0) &= \{\varepsilon\} && \implies |c_{\mathcal{L}}(0)| = 1 \\ c_{\mathcal{L}}(1) &= \{0, 2, 4, 6, 8\} && \implies |c_{\mathcal{L}}(1)| = 5 \\ c_{\mathcal{L}}(2) &= \{10, 12, 14, \ldots, 98\} && \implies |c_{\mathcal{L}}(2)| = 45 \\ &\vdots \end{aligned}$$

Thus, its profile would be 1, 5, 45, 450, 4500, ...

Definition 4.3. The **structure generating function** of a language is the generating function that describes its profile.

$$g_{\mathcal{L}}(z) = \sum_{n=0}^{\infty} z^n c_{\mathcal{L}}(n)$$

Example 4.2. Continuing with the language from Example 4.1 its structure generating function would be:

$$g_{\mathcal{L}}(z) = 1z^0 + 5z^1 + 45z^2 + 450z^3 + 4500z^4 + \cdots$$

Lemma 4.1.

$$g_{\mathcal{A}^*}(Z) = \sum_{n=0}^{\infty} |\mathcal{A}|^n z^n$$

Proof. By definition of the language $\mathcal{A}^*$ all combinations of letters from $\mathcal{A}$, no matter the length, is part of the language. So, by induction:
Base case: $n = 0$

The only word of length 0 in any and all languages is ε, so $c_{\mathcal{A}^*}(0) = 1 = |\mathcal{A}|^0$.
Inductive step: Assume that $c_{\mathcal{A}^*}(k) = |\mathcal{A}|^k$, consider the set of words of length $k + 1$.

The only way to construct one of these words is to append a letter from $\mathcal{A}$ to it. For each word of length k you can choose any letter to append so you have $|\mathcal{A}|$ choices. So $c_{\mathcal{A}^*}(k+1) = |\mathcal{A}| \times |\mathcal{A}|^k = |\mathcal{A}|^{k+1}$. □

Earlier we did set operations on languages and proved what that did to their regularity, but how do they affect the language's structure generating function?

Lemma 4.2. *The sum of the structure generating function of a language and that of its complement is the structure generating function of $\mathcal{A}^*$.*

$$g_{\mathcal{L}}(z) + g_{\mathcal{L}'}(z) = g_{\mathcal{A}^*}(z)$$

Proof. Let us substitute in the definition of g and the result of Lemma 4.1.

$$\begin{aligned} g_{\mathcal{L}}(z) + g_{\mathcal{L}'}(z) &= g_{\mathcal{A}^*}(z) \\ \sum_{n=0}^{\infty} z^n c_{\mathcal{L}}(n) + \sum_{n=0}^{\infty} z^n c_{\mathcal{L}'}(n) &= \sum_{n=0}^{\infty} |\mathcal{A}|^n z^n \\ \sum_{n=0}^{\infty} z^n c_{\mathcal{L}}(n) + z^n c_{\mathcal{L}'}(n) &= \sum_{n=0}^{\infty} |\mathcal{A}|^n z^n \\ \sum_{n=0}^{\infty} z^n (c_{\mathcal{L}}(n) + c_{\mathcal{L}'}(n)) &= \sum_{n=0}^{\infty} |\mathcal{A}|^n z^n \end{aligned}$$

In words, if the number of words of length n in $\mathcal{L}$ plus the number of word of length n not in $\mathcal{L}$ is the total number of words of length n, then we've proven the lemma. This is trivially true.

□

Lemma 4.3.

$$\alpha\mathcal{L} = \alpha w : w \in \mathcal{L}$$

Thus:

$$c_{\alpha\mathcal{L}}(n) = c_{\mathcal{L}}(n-1).$$

So:

$$g_{\alpha\mathcal{L}}(z) = z g_{\mathcal{L}}(z).$$

Lemma 4.4. *If $\mathcal{L}_1$ and $\mathcal{L}_2$ are languages on alphabet A that are disjoint then:*

$$g_{\mathcal{L}_1 \cup \mathcal{L}_2}(z) = g_{\mathcal{L}_1}(z) + g_{\mathcal{L}_2}(z)$$

5 The Structure of Automata Regular Languages

Google became the company that it is today because of their search engine which, originally, functioned on the PageRank algorithm. A simplified version of this algorithm works by creating a matrix, M, that encodes how many times a certain site, i, links to a certain other site, j[3] at $M_{i,j}$. This is seen as the importance of that site. Multiplying M by itself, repeatedly, propagates the importance of a site to the sites it links to, known as the Eigenvector Centrality of a graph. In other words, if the BBC links to your website, your website becomes more important and trustworthy, this then makes all sites you link to also more trustworthy. Do this for a certain number of iterations and you can sort your search results like Google used to.

Looking at our automata as a graph we could do this to see how many words are in each state after a certain number of iterations using matrix multiplication. Let us define the **transition matrix** of our automata first:

Definition 5.1. Given an automata $A = (S, S', \mathcal{I}, T)$, let $M_{i,j}$ be the number of paths (where each input letter is a path) from the state $i \in S$ to the state $j \in S$, in other words the number of arrows from i to j in the transition diagram. More directly $M_{i,j} = |\{\alpha \in \mathcal{I} : T(i, \alpha) = j\}|$.

So the **transition matrix** of A is $M = (M_{i,j})_{i,j \in S}$

Lemma 5.1.

$$\forall i \in S, \sum_{j \in S} M_{i,j} = |\mathcal{I}|$$

Proof. For any state $i \in S$, the sum of the row i, $\sum_{j \in S} M_{i,j}$, is the number of arrows leaving i. Which, as the transition function is complete, is one arrow for each possible input. □

Furthermore, we can show that, given a number of words at a certain state, we can use M to calculate which state those words go to when processing their next letter.

Lemma 5.2. *Sure?*

$$c_{\mathcal{L}(A,i)}(n) = \begin{cases} 0 & \text{if } n = 0 \text{ and } i \notin S' \\ 1 & \text{if } n = 0 \text{ and } i \in S' \\ \sum_{j \in S} c_{\mathcal{L}(A,j)}(n-1) M_{i,j} & \text{if } n > 0. \end{cases}$$

Proof. In words, this Lemma is stating that the number of words in the language defined by A, given the starting state i, that are of length n is the sum of the number of words of length $n-1$ for each state that you can get to from i in one transition.

Effectively, starting at i, you follow the transition for every possible input letter and ask how many words of length $n-1$ start from this state. This is similar to a breadth first search of a graph.

Base Case: $n = 0$ and $i \notin S'$

If i is not an accepting state there is no word of length 0. Hence, $c_{\mathcal{L}(A,i)}(0) = 0$.

Base Case: $n = 0$ and $i \in S'$

[3] In reality this is also scaled by a "trust factor".

If i is an accepting state there is one word of length 0, the empty word ε. Hence, $c_{\mathcal{L}(A,i)}(0) = 1$.
Base Case: $n = 1$

The words of length one we are considering here have only one symbol, w_0. w_0 is accepting if and only if it transitions into an accepting state, i.e. $T(i, w_0) \in S'$. The previous base cases mean that the word constructed of this symbol being accepting is equivalent to stating that $c_{\mathcal{L}(A,T(i,w_0))}(0) = 1$.

$\sum_{j\in S} M_{i,j}$ is by definition the number of paths of length 1 from state i to state j. So if the word leading to j is accepting it is a part of our language and we keep it, i.e. multiply it by $1 = c_{\mathcal{L}(A,j)}(0)$. If it is not accepting we ignore it by multiplying by $0 = c_{\mathcal{L}(A,j)}(0)$. So $c_{\mathcal{L}(A,i)}(1) = \sum_{j\in S} c_{\mathcal{L}(A,j)}(0)M_{i,j}$.
Inductive Step: Assume $c_{\mathcal{L}(A,i)}(k) = \sum_{j\in S} c_{\mathcal{L}(A,j)}(k-1)M_{i,j}$.

This can be done using the same logic as the case of $n = 1$ but instead of relying on the base cases we rely on the inductive step.

□

We can do this exact same "vectorisation" with our structure generating function. We work upwards instead this time which means we don't need to subtract one for each recursion, we simply multiply by z and keep going until infinity.

Lemma 5.3.

$$g_{\mathcal{L}(A,i)}(z) = c_{\mathcal{L}(A,i)}(0) + z\sum_{j\in S} g_{\mathcal{L}(A,i)}(z)M_{i,j}$$

We can make this more general by considering all states simultaneously.

Definition 5.2. Given:

$$v = c_{\mathcal{L}(A,i)}(0) = \begin{cases} 1, & \text{if } i \in S', \\ 0, & \text{if } i \notin S', \end{cases}$$

We call $(v_i)_{i\in S}$ (a column vector) the **acceptance vector**.

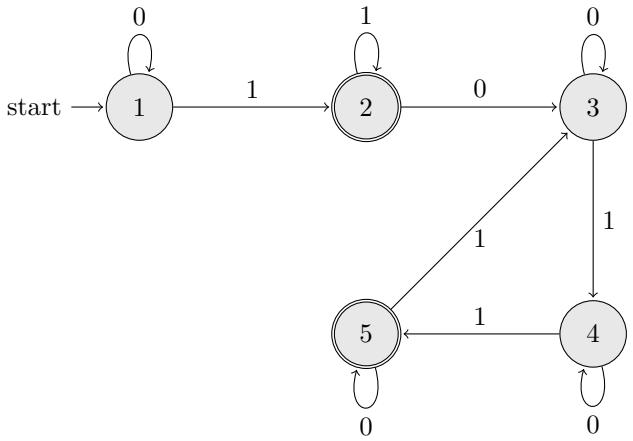
Theorem 5.1.

$$g(z) = v + zMg(z)$$

6 Structure From Automata

Our abstraction of language and our condition of automata regularity have come together rather beautifully to connect with graph theory and linear algebra. Let us explore what Theorem 5.1 allows us to do with an example.

Example 6.1. Consider the automata in Figure 6.1, a slightly more complex automata than we have looked at previously.



As the accepting states are 2 and 5 we can easily construct the *acceptance vector*.

$$v = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

We can also construct our *transition matrix* by observation. Where each state goes to itself once and the next one once, except the last state which goes to 3 instead.

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

These can be simply placed into Theorem 5.1 to give us the following equation.

$$g(z) = v + zMg(z)$$

$$\begin{bmatrix} g_1(z) \\ g_2(z) \\ g_3(z) \\ g_4(z) \\ g_5(z) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} + z \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} g_1(z) \\ g_2(z) \\ g_3(z) \\ g_4(z) \\ g_5(z) \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} + z \begin{bmatrix} g_1(z) + g_2(z) \\ g_2(z) + g_3(z) \\ g_3(z) + g_4(z) \\ g_4(z) + g_5(z) \\ g_3(z) + g_5(z) \end{bmatrix}$$

$$\begin{bmatrix} g_1(z) \\ g_2(z) \\ g_3(z) \\ g_4(z) \\ g_5(z) \end{bmatrix} = \begin{bmatrix} z(g_1(z) + g_2(z)) \\ 1 + z(g_2(z) + g_3(z)) \\ z(g_3(z) + g_4(z)) \\ z(g_4(z) + g_5(z)) \\ 1 + z(g_3(z) + g_5(z)) \end{bmatrix}$$

As our starting states is 1 we will use this to find g_1, however, this works for any starting state. By going row by row we get:

$$\begin{aligned} g_1(z) &= zg_1(z) + zg_2(z) \implies \frac{1-z}{z} g_1(z) = g_2(z) \\ g_2(z) &= 1 + zg_2(z) + zg_3(z) \implies \frac{1-z}{z} g_2(z) - 1 = g_3(z) \\ g_3(z) &= zg_3(z) + zg_4(z) \implies \frac{1-z}{z} g_3(z) = g_4(z) \\ g_4(z) &= zg_4(z) + zg_5(z) \implies \frac{1-z}{z} g_4(z) = g_5(z) \\ g_5(z) &= 1 + zg_3(z) + zg_5(z) \implies \frac{1-z}{z} g_5(z) - 1 = g_3(z) \end{aligned}$$

We can keep substituting these into the last equation to put it in terms of $g_1(z)$:

$$\begin{aligned} (1-z)g_5(z) - 1 &= zg_3(z) \\ (1-z)\left(\frac{1-z}{z} g_4(z) - 1\right) &= z\left(\frac{1-z}{z} g_2(z) - 1\right) \\ \frac{(1-z)^2}{z} g_4(z) - (1-z) &= (1-z)g_2(z) - z \\ &\vdots \longrightarrow \text{Full working available in the Appendix} \\ g_1(z) &= \frac{1 - 3z + 3z^2 - 2z^3 + z^6 - z^7}{z^2 - 5z^3 + 10z^4 - 11z^5 + 7z^6 - 2z^7} \end{aligned}$$

This is an example of a Theorem we won't cover here, which is that: for any automata and starting state, the structure generating function will always be a ratio of polynomials.

A Extended Working

$$
\begin{aligned}
(1-z)g_5(z) - 1 &= zg_3(z) \\
(1-z)\left(\frac{1-z}{z}g_4(z) - 1\right) &= z\left(\frac{1-z}{z}g_2(z) - 1\right) \\
\frac{(1-z)^2}{z}g_4(z) - (1-z) &= (1-z)g_2(z) - z \\
\frac{1-z}{z}g_4(z) - 1 &= g_2(z) - \frac{z}{1-z} \\
\frac{1-z}{z}\frac{1-z}{z}g_3(z) - 1 &= \frac{1-z}{z}g_1(z) - \frac{z}{1-z} \\
\frac{(1-z)^2}{z^2}g_3(z) - 1 &= \frac{1-z}{z}g_1(z) - \frac{z}{1-z} \\
\frac{(1-z)^2}{z^2}\left(\frac{1-z}{z}g_2(z) - 1\right) - 1 &= \frac{1-z}{z}g_1(z) - \frac{z}{1-z} \\
\frac{(1-z)^3}{z^3}g_2(z) - \frac{(1-z)^2}{z^2} - 1 &= \frac{1-z}{z}g_1(z) - \frac{z}{1-z} \\
\frac{(1-z)^3}{z^3}\frac{1-z}{z}g_1(z) - \frac{(1-z)^2}{z^2} - 1 &= \frac{1-z}{z}g_1(z) - \frac{z}{1-z} \\
\frac{(1-z)^4}{z^4}g_1(z) - \frac{(1-z)^2}{z^2} - 1 &= \frac{1-z}{z}g_1(z) - \frac{z}{1-z} \\
g_1(z)\left(\frac{(1-z)^4}{z^4} - \frac{1-z}{z}\right) &= \frac{(1-z)^2}{z^2} - \frac{z}{1-z} + 1 \\
g_1(z)\frac{(1-z)^4 - z^3(1-z)}{z^4} &= \frac{(1-z)^3 - z^3}{z^2(1-z)} + 1 \\
g_1(z) &= \frac{\frac{z^4((1-z)^3 - z^3)}{z^6(1-z)} + z^4}{(1-z)^4 - z^3(1-z)} \\
g_1(z) &= \frac{z^4((1-z)^3 - z^3) + z^{10}(1-z)}{z^6(1-z)^5 - z^9(1-z)^2} \\
g_1(z) &= \frac{z^4 - 3z^5 + 3z^6 - 2z^7 + z^{10} - z^{11}}{z^6 - 5z^7 + 10z^8 - 11z^9 + 7z^{10} - 2z^{11}} \\
g_1(z) &= \frac{1 - 3z + 3z^2 - 2z^3 + z^6 - z^7}{z^2 - 5z^3 + 10z^4 - 11z^5 + 7z^6 - 2z^7}
\end{aligned}
$$