LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

CHAIR OF THEORETICAL COMPUTER SCIENCE AND THEOREM PROVING

Formal Languages and Automatas

Ruben Triwari

Seminar Paper on Formal Languages and Automata
for the course “Algebra & Computer Science”

Supervisor: Prof. Dr. Jasmin Blanchette

Advisor: Xavier Genereux

Submission Date: August 16, 2024

Disclaimer

I confirm that this seminar paper is my own work and I have documented all sources and material used.

Munich, August 16, 2024 Author

Contents

1 Introduction

This paper aims to give a gentle introduction to the world of automata with formal power series. Formal power series are really useful for weighted automata, but not so much for the topics we are covering in this paper. Thus, some proofs may not be as simple as in other literature. First, we will go over some definitions, starting from scratch, but ending up with the definition of an automaton using a matrix and formal power series. In the third chapter, we will provide some intuition as to why one would use a formal power series in this context and show that the definition of formal language using formal power series and sets is equivalent. At the end of the third chapter, we will discuss why formal power series are so powerful in the context of automata theory. In the fourth chapter, we will prove the kleene's theorem using the formal power series representation of sets. At the end, we will give an outlook on weighted automata.

2 Backround

Firstly, we need to define some structures and ideas, that will be really important in this paper. We will define formal languages in two ways and then the automata accordingly. A formal language is just a set of words, where words are a series of characters from an alphabet

Definition 1 *An Alphabet is an abritrary non empty finite set denoted with* Σ

Example 2.1 *Simple examples for Alphabets would be:*

1. $\Sigma = \{x\}$
2. $\Sigma = \{a,b,c,d,\dots z\}$
3. $\Sigma = \{1,2,3,\dots n\}$ *where* $n \in \mathbb{N}$

This definition is rather general, because an alphabet Σ can really be anything. We could also define an alphabet as a subset of the natural numbers $\Sigma \subset \mathbb{N}$. This is equivalent to the previous definition, because we can always find a one-to-one map between a finite amount of objects and the natural numbers. In other words we can always enumerate a finite amount of objects. The above definition results in more readable words, thus resulting in more readable problems. Furthermore, it's unintuitive to have words full of numbers and languages full of number series. That is why we are choosing this definition, like most of the literature.
Now that our first building block is defined, we can define words, which are just a series of characters in an alphabet.

Definition 2 *Let* Σ *be an fixed alphabet, then w is a series of characters out of* Σ*.*

$$w = (a_1, a_2, \dots, a_n) \in \Sigma^n$$

We will denote it by:

$$w = a_1 a_2 \dots a_n$$

The empty word which contains no characters, and is denoted by ε*.*

Example 2.2 *Simple examples for words would be:*

1. $\Sigma = \{a,b,c\} \rightsquigarrow w_1 = abc$
2. $\Sigma = \{x\} \rightsquigarrow w_2 = xxxx$
3. $\Sigma = \{1,2,3\} \rightsquigarrow w_3 = 112233$
4. $\Sigma = \{1,2,3\} \rightsquigarrow w_4 = \varepsilon$

The empty word ε is analogous to the empty set in set theory. It has a lot of uses, but one obvious one is that now one can define a monoid over words $(M, \cdot, \Sigma, \varepsilon)$. M is a set of words and multiplication is the concatenation of words, then ε is the neutral element of the monoid. Which brings us to the next definition.

Definition 3 *Let Σ be an fixed alphabet and w, v words with characters out of Σ. Then the contenation is defined by:*

$$w \cdot v := wv$$

sometimes just written as wv. With powers recursively defined by:

$$w^0 := \varepsilon$$
$$w^1 := w$$
$$w^{n+1} := w^n \cdot w$$

In simple terms, a concatenation of two words is just a new word with the two words were chained together. Now we can again build on words to define languages.

Definition 4 *Let Σ be an fixed alphabet, then a formal language is a set of words, with characters in Σ.*

Example 2.3 *Simple examples for formal languages would be:*

1. $\Sigma = \{a,b,c\} \rightsquigarrow L = \{aaa, bbb, ccc, abc\}$
2. $\Sigma = \{x\} \rightsquigarrow L = \{x, xx, xxx, xxxx\}$
3. $\Sigma = \{1,2,3\} \rightsquigarrow L = \{11, 22, 33, 123, \varepsilon\}$

This definition is really clear and simple. We will see a less intuitive way to define language shortly. Obviously, the simpler one has some downsides, which we will discuss later in the paper.
The kleene star * is a really powerful operator, to define it, we need to first define contention for languages.

Definition 5 *Let Σ be an fixed alphabet and L, V languages with words constructed out of Σ. Then the contenation of formal languages is defined by:*

$$L \cdot V := \{l \cdot v | l \in L \wedge v \in V\}$$

sometimes just written as LV. With powers defined recursively:

$$\begin{aligned} L^0 &:= \{\varepsilon\} \\ L^1 &:= L \\ L^{i+1} &:= \{l_i \cdot l | l_i \in L^i \wedge l \in L\} \end{aligned}$$

In simple terms, we are creating a new language by combining every word of the first language with every word of the second. Now we are able to define the kleene star operator

Definition 6 *Let Σ be an fixed alphabet and L a language with words constructed out of Σ. Then the kleene star of a formal language is defined by:*

$$L^* := \bigcup_{i \in \mathbb{N}_0} L^i = L^0 \cup L^1 \cup L^2 \cup \ldots$$

and

$$L\hat{} := \bigcup_{i \in \mathbb{N}} L^i = L^1 \cup L^2 \cup L^3 \cup \ldots$$

In simple terms, it generates all possible permutations of arbitrary length of a formal language. This is really powerful, because Σ^* is the biggest language one can construct from the alphabet Σ. Thus, every language L generated by Σ is just a subset of Σ^*.
Unintuitively, we can also write a formal language as a formal power series.

Definition 7 *Let Σ be an fixed alphabet, then a formal language can be written as formal power series L:*

$$L : \Sigma^* \to \mathbb{B}$$

where

$$\mathbb{B} = \{0, 1\}$$

$$L = \sum_{w \in \Sigma^*} (L, w) w \ \texttt{where} \ (L, w) \in \mathbb{B}.$$

Note: *The set of formal power series is denoted by $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle = \{\Sigma^* \to \mathbb{B}\}$*
Notation: *If the language consist only out of one character or the empty word $a \in \Sigma \cup \{\varepsilon\}$, then the formal power series would be:*

$$L = h \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$$

where

$$h : \Sigma^* \to \mathbb{B}$$

$$h(w) = \begin{cases} 1 & \text{if } w = a \\ 0 & \text{otherwise} \end{cases}.$$

In this case we just write $L = a$. The language with no words is denoted by $L = 0$.

This looks like a sum, we call it a series, but it is actually a map from words to zero or one. The sum is a way to write a function definition: In front of every word $w \in \Sigma^*$ in the series, there is a $x \in \mathbb{B}$, which is zero or one. So we can interpret every element in the series as an assignment: $w \mapsto x$. All words that map to one are in the language, the others are not. So we can write a formal language as a function $\Sigma^* \to \mathbb{B}$.
The representation with sets has the advantage that a formal language will inherit all operations on sets like complement, union, and intersection. Those operations are really useful, so we will define similar ones on the power series version.

Definition 8 *Let Σ be an fixed alphabet and $L, V \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$, where*

$$L = \sum_{w \in \Sigma^*} (L, w)w \quad \textit{and} \quad V = \sum_{w \in \Sigma^*} (V, w)w.$$

Addition and multiplication on formal power series is defined as follows:

$$L + V := \sum_{w \in \Sigma^*} ((L, w) + (V, w))w \quad \textit{and} \quad L \cdot V := \sum_{\substack{u,v \in \Sigma^* \\ w = uv}} ((L, u) \cdot (V, v))w$$

This definition is not recursive, because the right addition and muliplication operater are from the Boolean Algebra $\mathbb{B} = \{0, 1\}$.

$+$	*0*	*1*
0	*0*	*1*
1	*1*	*1*

$\times$	*0*	*1*
0	*0*	*0*
1	*0*	*1*

With powers defined recursively:

$$\begin{aligned} L^0 &:= \varepsilon \\ L^1 &:= L \\ L^{i+1} &:= \sum_{\substack{u,v \in \Sigma^* \\ w = uv}} ((L^i, u) \cdot (L, v))w, \end{aligned}$$

Kleene Star can now also be defined for formal power series:

$$L^* := L^0 + L^1 + L^2 + L^3 + \cdots = \sum_{i \in \mathbb{N}_0} L^i.$$

and

$$L^{\hat{}} := L^1 + L^2 + L^3 + \cdots = \sum_{i \in \mathbb{N}_0} L^i.$$

With these operation defined our two versions of formal languages are powerful tools, which will be compared in the following sections. In order to conpare them we will also need finite automatas. Simillar to formal languages there are multiple ways one can define finite automatas. In this paper we will define two. One is more suitable for combining it with the set version of formal languages and the other one is more siutable for the formal power series representation. First we will define the one more suitable for the set version.

Definition 9 *A finite automata is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where*

1. *Q is a finite set of states,*
2. *Σ is an alphabet, all inputs are constructed from,*
3. *$\delta : Q \times \Sigma \to Q$ is the transition function, carrying the automata from one state to another,*
4. *$q_0 \in Q$ is the initial state of the automata,*
5. *$F \subset Q$ are the accepting states of the automata.*

In simple terms, a finite automaton gets an input string and a starting state. Then it works through the input string, starting at the first character and carrying the starting state to another according to the transition function. Exactly the same procedure applies for the second input string character and the remaining characters. If this process ends up in an accepting state, the automata accept the input string. Consequently, the finite automata generate a language, namely the words that the automata accepts. The definition above is one possible way to encode a finite automaton. Like with graphs, we can encode the transition function of a finite automata, with just a matrix.

Definition 10 *Let $M = (Q, \Sigma, \delta, q_0, F)$ be an abitrary finite automata, where $Q = \{s_1, s_2, \dots s_n\}$. Frthermore, we need a helper function:*

$$\delta' : \mathbb{N} \times \mathbb{N} \to \mathbb{B}\langle\langle \Sigma^* \rangle\rangle$$

where

$$\delta'(i,j) = \begin{cases} \alpha, & \text{if } \exists \alpha \in \Sigma : \delta(s_i, \alpha) = s_j \\ 0, & \text{otherwise} \end{cases}.$$

Then the finite automata M can be encoded with a Matrix $A \in \mathbb{B}\langle\langle \Sigma^* \rangle\rangle^{n \times n}$ in the following way:

$$A = \begin{pmatrix} \delta'(1,1) & \delta'(1,2) & \delta'(1,3) & \dots & \delta'(1,n) \\ \delta'(2,1) & \delta'(2,2) & \delta'(2,3) & \dots & \delta'(2,n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \delta'(n,1) & \delta'(n,2) & \delta'(n,3) & \dots & \delta'(n,n) \end{pmatrix}.$$

We can denote a state transition $s_i \to s_j$ with $s_i A_{ij} = s_j$.
Note: *We definined $A \in \mathbb{B}\langle\langle \Sigma^* \rangle\rangle^{n \times n}$ and not $A \in \Sigma^{n \times n}$, because otherwise the defintion is not closed under matrix multiplication, nor matrix addition.*

The kleene star of a Matrix can intuively be defined as:

$$A^* = E_n + A + A^2 + A^3 \cdots = \sum_{i \in \mathbb{N}_0} A^i$$

and

$$A\hat{} = A + A^2 + A^3 \cdots = \sum_{i \in \mathbb{N}} A^i$$

where

$$E_n \in \mathbb{B}\langle\langle \Sigma^* \rangle\rangle^{n \times n}$$

$$(E_n)_{ij} := \begin{cases} \varepsilon & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \in \mathbb{B}\langle\langle \Sigma^* \rangle\rangle$$

Now, we can write our finite automata in a shorter way:

$$M' = (\Sigma, A, q_0, F)$$

This definition only works, because we have finite sets. Note that we can omit the state set, since, as mentioned above, there always exists a one-to-one map between a set with finite objects and a subset of the natural numbers. The first object gets the number one, the second the number two, and so forth. The new definition of the state transition function works like a lookup table, when one wants to know what characters map state i to state j, it is a simple lookup: M_{ij}.

Example 2.4 *Here is a simple automata depicted, where the circles are states , the arrows are transitions, the start is indicating the initial state, and the double lined circle is the accepting state.*

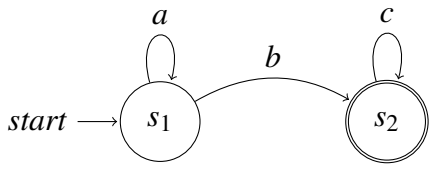
$$M_{tuple} = (\{s_1, s_2\}, \{a, b, c\}, \delta_0, s_1, \{s_2\}), M_{matrix} = (\{a, b, c\}, A, s_1, \{s_2\})$$

where

$$\delta_0(s_1, a) = s_1, \;\; \delta_0(s_1, b) = s_2, \;\; \delta_0(s_2, c) = s_2$$

and

$$A = \begin{pmatrix} a & b \\ 0 & c \end{pmatrix} \in \mathbb{B}\langle\langle \Sigma^* \rangle\rangle^{2 \times 2}.$$



The resulting accepting formal language is:

$$\textit{Set represntation: } L = \{a^n b c^n : n \in \mathbb{N}\}$$

$$\textit{Formal power series represntation: } L = a^* \cdot b \cdot c^* = a^* b c^*$$

3 Formal Languages: Sets vs. Formal Power Series

In this chapter, we will discuss the obvious differences and similarities between both representations. First, we will give an intuition as to why one might use a series to represent a function definition. Secondly, we will map the formal power series version into the set version and the other way around, to get a better understanding of both. Lastly, we will discuss why formal power series are more powerful than just sets.
The intuition behind the notation of formal power series is quite simple. The operation defined on the formal power series, which are maps, are really close to a series. In addition, formal power series and Normal series are quite similar, namely defined pointwise.
In the following are $L,V \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ and A,B series.

$$L+V = \sum_{w\in\Sigma^*}((L,w)+(V,w))w \quad \leftrightarrow \quad A+B = \sum_{n\in\mathbb{N}}(a_n+b_n)x^n$$

The multiplication defined on formal power series are again really close to the Cauchy product of series. In the following are $L,V \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ and A,B series.

$$L\cdot V = \sum_{\substack{u,v\in\Sigma^* \\ w=uv}}((L,u)\cdot(V,v))w \quad \leftrightarrow \quad A\cdot B = \sum_{\substack{i,j\in\mathbb{N} \\ n=i+j}}(a_i\cdot b_j)x^n$$

Consequently, it is quite handy to use a formal power series as a function definition, because the operations seem natural with series. Despite the notation being quite confusing at first glance.
Next, we will show that both representations of a formal language are equivalent, by mapping one to another. To be sure that both representations are actually equivalent, we need to show there exists a bijective map between the formal power series and the set of words. This map needs to preserve the structure, which means the order of function application and operator application must be irrelevant. Such maps are called isomorphisms, and the structures for that an isomorphism exist are called isomorphic to each other. Thus, we want to show that both representation are isomorphic.

Proof: Let Σ be an fixed alphabet, then we claim that:

$$\mathbb{B}\langle\langle\Sigma^*\rangle\rangle \cong \mathscr{P}(\Sigma^*)$$

In order to show those are isomorphic, we provide a map $\phi : \mathbb{B}\langle\langle\Sigma^*\rangle\rangle \to \mathscr{P}(\Sigma^*)$ and its inverse $\phi^{-1} : \mathscr{P}(\Sigma^*) \to \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ and show that ϕ is bijective and preservers structure, formally:
Let $L,V \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$, then for ϕ must hold:

$$\phi(L+V) = \phi(L)\cup\phi(V) \tag{1}$$
$$\phi(L\cdot V) = \phi(L)\cdot\phi(V) \tag{2}$$
$$\phi(L^*) = \phi(L)^* \tag{3}$$

We define ϕ and its inverse as follows:

$$\phi(L) = \{w\in\Sigma^* : (L,w) = 1\}$$

$$\phi(L)^{-1} = h_L$$

where

$$h_L : \Sigma^* \to \mathbb{B}\langle\langle \Sigma^* \rangle\rangle$$

$$h_L(w) = \begin{cases} 1, & \text{if } w \in L \\ 0, & \text{otherwise} \end{cases}$$

To prove that ϕ is bijective, it must hold

$$\phi^{-1}(\phi(L)) = L$$

and

$$\phi(\phi^{-1}(L)) = L.$$

First we show $\phi^{-1}(\phi(L)) = L$
Let $L \in \mathbb{B}\langle\langle \Sigma^* \rangle\rangle$ abitrary but fixed, with

$$L = \sum_{w \in \Sigma^*} (L, w)w.$$

Then

$$\begin{aligned} \phi^{-1}(\phi(L)) = \phi^{-1}(\{w \in \Sigma^* : (L,w) = 1\}) &= \begin{cases} 1 & \text{if } w \in \{w \in \Sigma^* : (L,w) = 1\} \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} 1 & \text{if } (L,w) = 1 \\ 0 & \text{otherwise} \end{cases} \\ &= \sum_{w \in \Sigma^*} (L,w)w = L. \end{aligned}$$

Next we need to show that $\phi(\phi^{-1}(L)) = L$.
Let $L \in \mathscr{P}(\Sigma^*)$ abitrary but fixed, then

$$\phi(\phi^{-1}(L)) = \phi(h_L) = \{w \in \Sigma^* : (h_L, w) = 1\} = \{w \in \Sigma^* : h_L(w) = 1\} = \{w \in \Sigma^* : w \in L\} = L.$$

Consequently, ϕ is a one-to-one correspondence between formal power series and set of words. We are not done yet, we still need to prove that ϕ preserves structure, thus we need to show $(1), (2)$, and (3). Starting with number one:

$$\textbf{(1)} \quad \phi(L+V) = \phi(L) \cup \phi(V)$$

In the following let $L, V \in \mathbb{B}\langle\langle \Sigma^* \rangle\rangle$ arbitrary but fixed. Then

$$\begin{aligned} \phi(L+V) &= \{w \in \Sigma^* : (L,w) + (V,w) = 1\} \\ &= \{w \in \Sigma^* : (L,w) = 1 \vee (V,w) = 1\} \\ &= \{w \in \Sigma^* : w \in \{x \in \Sigma^* : (L,x) = 1\} \vee w \in \{x \in \Sigma^* : (V,x) = 1\}\} \\ &= \{w \in \Sigma^* : w \in \phi(L) \vee w \in \phi(V)\} = \phi(L) \cup \phi(V). \end{aligned}$$

Continuing with two:

$$\textbf{(2)} \quad \phi(L \cdot V) = \phi(L) \cdot \phi(V)$$

In the following let $L, V \in \mathbb{B}\langle\langle \Sigma^* \rangle\rangle$ arbitrary but fixed. Then

$$\begin{aligned}
\phi(L \cdot V) &= \{uv \in \Sigma^* : (L,u) \cdot (V,v) = 1\} \\
&= \{uv \in \Sigma^* : (L,u) = 1 \wedge (V,v) = 1\} \\
&= \{uv \in \Sigma^* : u \in \{w \in \Sigma^* : (L,w) = 1\} \wedge v \in \{w \in \Sigma^* : (V,w) = 1\}\} \\
&= \{uv \in \Sigma^* : u \in \phi(L) \wedge v \in \phi(V)\} = \phi(L) \cdot \phi(V).
\end{aligned}$$

Last but not least, three:

$$\textbf{(3)} \quad \phi(L^*) = \phi(L)^*$$

In the following let $L \in \mathbb{B}\langle\langle \Sigma^* \rangle\rangle$ arbitrary but fixed. Then

$$\phi(L^*) = \phi(\sum_{i \in \mathbb{N}_0} L^i) \overset{\textbf{(1)}}{=} \sum_{i \in \mathbb{N}_0} \phi(L^i) \overset{\textbf{(2)}}{=} \sum_{i \in \mathbb{N}_0} \phi(L)^i = \phi(L)^*$$

Thus ϕ is an isomorphism and our claim follows. □

Now we have shown that both definitions are equivalent, but why should one use the less natural formal power series definition? The answer to this, as so often in mathematics, is **abstraction**. Finite automata are accepting formal languages, thus we can say they generate a formal language. When abstracting from finite automata's to weighted automata's, which have a weight or cost between each transition, the generated object is not longer a language, but a map that assigns a weight or cost to each word. Generally, this weight or cost can be seen as a semi-ring (a ring that does not necessarily have an additive inverse), which are objects that behave like numbers. We will denote an arbitrary semi-ring with $\mathbb{S}$. Those maps, that get accepted by a weighted automata, have the signature $L : \Sigma^* \to \mathbb{S}$ and are indeed, a formal power series over $\mathbb{S}$, consequently $L \in \mathbb{S}\langle\langle \Sigma^* \rangle\rangle$. To summarize, finite automata generate formal power series over $\mathbb{B}$ and weighted automata generate formal power series over $\mathbb{S}$. So formal power series are, in fact, an abstraction of formal languages.

4 Kleene's theorem (Power series, Matrix Automata)

In this chapter, we will examine, what kinds of languages get accepted by a finite automaton. In order to do that, we will prove that all languages that get accepted by an finite automaton are rational, and every rational language can be generated by a finite automaton (kleene's theorem). A rational language is defined inductively.

Definition 11 *Let Σ be an fixed alphabet, then a rational language is defined as follows:*

1. *A language $L = a \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ with a single character $a \in \Sigma$ is said to be rational.*
2. *The empty word ε is rational.*
3. *The sum of two rational Languages $L_1, L_2 \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ are again rational $L = L_1 + L_2$.*
4. *The product of two rational Languages $L_1, L_2 \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ are again rational $L = L_1 \cdot L_2$.*
5. *The kleene star of a rational language $L \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ is again rational $V = L^*$.*

So we start with just one character and then building with our operations step by step a language. To show kleene's theorem, we have to prove two directions. First, we will prove that finite automata only accept rational languages. Afterward, we will prove that every rational language is accepted by a finite automaton. In an effort to do that, we need to understand certain properties of the matrix encoding of finite automata. Raising the power of two of a Matrix is defined by:
Let $A \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{n\times n}$ be an arbitrary matrix, then

$$(A^2)_{ij} = \sum_{k=1}^{n} A_{ik}A_{kj}.$$

As mentioned before, this matrix is a transition function of a finite automata, where we can look up what character carries one state in another. In A_{ij} is the character that carries i to j, thus multiplying A by itself gives us a new matrix with all words of length two, given as input, and i is the starting node that would carry i to j in two steps. In the formula above, we see that we are trying to go through all nodes $k \in \{1, 2, 3, \dots n\}$ to end up in j and then uniting them together with addition. Generally, we can have a matrix $A \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{n\times n}$ risen to the power of $n \in \mathbb{N}$:

$$(A^n)_{ij} = \sum_{k_1=1}^{n}\sum_{k_2=1}^{n}\cdots\sum_{k_{n-1}=1}^{n} a_{ik_1}a_{k_1k_2}a_{k_2k_3}\dots a_{k_{n-1}j}$$

This is not beautiful at all, but the idea stays exactly the same: now we have in $(A^n)_{ij}$ all words that carry i to j in n steps, brute forcing through all possible paths. Now to get all possible paths of arbitrary length, we just add them all together:

$$E_n + A + A^2 \cdots = \sum_{i\in\mathbb{N}} A^i = A^*$$

Yielding, that in $(A^*)_{ij}$ are all possible words, carrying the automata from i to j. If i is the starting node and j the accepting node, the accepting language of the automata is just $(A^*)_{ij}$. That means, A^* stores all accepting languages of an automaton, with transition matrix A. With those important properties, we can now move on to the first prove.

Proof:

claim: All finite automata only accept rational languages.

Let $A \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{n\times n}$ be a transition function of an arbitrary finite automata. A is rational because the transition function consists only out of characters. Since A^* stores every accepting language of that finite automata, for every possible initial state and accepting state, we just need to prove that all entries in A^* are rational languages. We will prove this statement via induction over the number of states, namely the dimension of the square matrix $n \in \mathbb{N}$.
Base case: $n = 1$
Thus

$$A \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{1\times 1} = \mathbb{B}\langle\langle\Sigma^*\rangle\rangle.$$

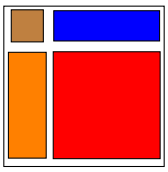
We know that $A \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ is rational, but that also means that A^* is per definition rational, which finishes the base case.
Induction step: $n \to n+1$
We can assume the claim holds for $n \in \mathbb{N}$ and need to prove it also holds for $n+1$. Thus, we can assume that a transition matrix with dimension $n \times n$ has only rational entries.
Let $A \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{(n+1)\times(n+1)}$ be a transition function of an arbitrary finite automata. Our goal is to reduce the square matrix A by one dimension. In order to do that, we divide the matrix into four blocks:

$$A = \begin{pmatrix} A(1,1) & A(1,2) \\ A(2,1) & A(2,2) \end{pmatrix}$$

where

$$A(1,1) \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{1\times 1}, A(2,1) \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{n\times 1}, A(1,2) \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{1\times n}, A(2,2) \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{(n-1)\times(n-1)}$$

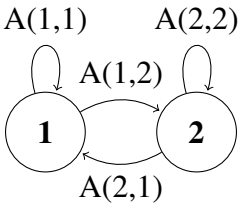
visually:



We can interpret our block matrix as a transition matrix with four entries, thus having two states. Our new two states consist of the original states in the following way:

$$\mathbf{1} = \{1\} \text{ and } \mathbf{2} = \{2,3,4,\ldots n+1\}$$

We can visulize the resulting automata:



Now, instead of calculating A^* with algebra, we can also just look at the automata and ponder what all possible paths are for $1 \to 1$, $1 \to 2$, $2 \to 1$, and $2 \to 2$. Note that the automaton is symmetrical, consequently we just need to find paths for $1 \to 1$ and $1 \to 2$. Starting with $1 \to 1$, we want to find all possible paths from state one to one with an arbitrary length. Starting at state one, we can either directly loop back to one, or we transition to two, then loop in two arbitrary amount of times and then transition back to one. This whole process can also be repeated an arbitrary amount of times, yielding the following formula:

$$A(1,1)^* = (A(1,1) + A(1,2)A(2,2)^*A(2,1))^*$$

Now, because of symmetry:

$$A(2,2)^* = (A(2,2) + A(2,1)A(1,1)^*A(1,2))^*$$

For $1 \to 2$, we can loop in one and then transition to two, in two We can either loop there for an arbitrary amount of time or transition to one loop in one an arbitrary amount of times and then go back to two. This process can also be repeated an arbitrary amount of times.

$$A(1,2)^* = A(1,1)^*A(1,2)(A(2,2)^* + A(2,1)A(1,1)^*A(1,2))^*$$

Again because of symmetry:

$$A(2,1)^* = A(2,2)^*A(2,1)(A(1,1)^* + A(1,2)A(2,2)^*A(2,1))^*$$

The transition matrix A^* has only rational entries, if all blocks matrices only have rational entries. The block matrices consist of, just entries of A that are rational, namely $A(1,1)$, $A(2,2)$, $A(1,2)$ and $A(2,1)$. As well as square matrices, which have a lower dimension than $n+1$, thus the kleene stars of those are also rational because of the induction assumption, namely $A(2,2)^*$, $A(1,1)^*$, $(A(1,1)^* + A(1,2)A(2,2)^*A(2,1))^*$, $(A(2,2)^* + A(2,1)A(1,1)^*A(1,2))^*$, $(A(2,2) + A(2,1)A(1,1)^*A(1,2))^*$ and $(A(1,1) + A(1,2)A(2,2)^*A(2,1))^*$. Thus, A^* has only rational entries, which finishes the proof. $\square$

Now, we have to prove the other direction:

Proof:

claim: Every rational language can be generated by an finite automata.

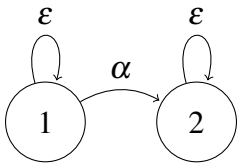
Previously in this chapter, we defined rational language inductively, naturally we want to prove the claim with structural induction. That means we need to construct an automata generating...

1. ... a language with one letter.
2. ... the product of two languages.
3. ... the sum of two languages.
4. ... the kleene star of one language.

In the following, all constructed automata will have the first state as initial state and all states as accepting state. Thus, the sum of the first row of the transition matrix A^* will be the accepted language. We begin to prove that **(1)** there are an automaton accepting a single letter.
Let Σ be an arbitrary but fixed alphabet and $\alpha \in \Sigma$ a letter, then:

$$A = \begin{pmatrix} \varepsilon & \alpha \\ 0 & \varepsilon \end{pmatrix}$$

Looking at the automata and ponder again, like in the previous proof, we see that $A = A^*$.



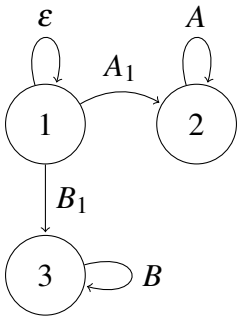
With our convention, we get the accepted language $L = \varepsilon + \alpha$, thus $\alpha \in L$.
Next, we will look at the sum of two rational language **(2)**. Let $A, B \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{n\times n}$ transition matrices of rational languages, then

$$B = \begin{pmatrix} \varepsilon & A_1 & B_1 \\ 0 & A & 0 \\ 0 & 0 & B \end{pmatrix}$$

where

$$A_1 = (a_{11}, a_{12}, a_{13}, \ldots a_{1n}) \text{ and } B_1 = (b_{11}, b_{12}, b_{13}, \ldots b_{1n}).$$

Depicting the automata:



With the same reasoning as above we get:

$$B^* = \begin{pmatrix} \varepsilon & A_1A^* & B_1B^* \\ 0 & A^* & 0 \\ 0 & 0 & B^* \end{pmatrix}$$

where

$$A_1A^* = (\sum_{k=1}^{n} a_{1k}a_{k1}^*, \sum_{k=1}^{n} a_{1k}a_{k2}^*, \sum_{k=1}^{n} a_{1k}a_{k3}^*, \dots \sum_{k=1}^{n} a_{1k}a_{kn}^*) = \hat{A_1}$$

and

$$B_1B^* = (\sum_{k=1}^{n} b_{1k}b_{k1}^*, \sum_{k=1}^{n} b_{1k}b_{k2}^*, \sum_{k=1}^{n} b_{1k}b_{k3}^*, \dots \sum_{k=1}^{n} b_{1k}b_{kn}^*) = \hat{B}_1.$$

Yielding the first row sum $L = \varepsilon + \hat{A_1} + \hat{B}_1$, which is exactly the sum of the accepted language of A and B.

Next, we will look at the product of two rational language **(3)**. Let $p,q \in \mathbb{N}$, $A \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{p\times p}$ and $B \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{q\times q}$ transition matrices of rational languages, then

$$C = \begin{pmatrix} A & PQ \\ 0 & B \end{pmatrix}$$

where

$$P \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{p\times q},\ Q \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{q\times p}$$

$$P_{ij} = \begin{cases} \varepsilon & \text{if } \ j = 1 \\ 0 & \text{otherwise} \end{cases}, \qquad Q_{ij} = \begin{cases} \varepsilon & \text{if } \ j = 1 \wedge i = 1 \\ 0 & \text{otherwise} \end{cases}$$

Using the formula of the last proof we get:

$$C^* = \begin{pmatrix} A^* & A^*PQB^* \\ 0 & B^* \end{pmatrix}$$

where

$$(A^*P)_{ij} = \begin{cases} \sum_{k=1}^{p} a_{1k}^* p_{k1} = \sum_{k=1}^{p} a_{1k}^* & \text{if} \quad i = 1 \wedge j = 1 \\ \sum_{k=1}^{p} a_{ik}^* p_{k1} = \sum_{k=1}^{p} a_{ik}^* & \text{if} \quad i > 1 \wedge j = 1 \\ 0 & \text{otherwise} \end{cases}$$

and

$$(QB^*)_{ij} = \begin{cases} \sum_{k=1}^{p} q_{1k} b_{kj}^* = b_{1j}^* & \text{if} \quad i = 1 \\ 0 & \text{otherwise} \end{cases}$$

We are only interested in the first row:

$$(A^*PQB^*)_{1j} = \sum_{k=1}^{q} (A^*P)_{1k} (QB^*)_{kj} = (A^*P)_{11} (QB^*)_{1j} = (\sum_{k=1}^{p} a_{1k}^*) b_{1j}^*$$

Summing the whole row of C^* together to get the accepted language:

$$L = (\sum_{k=1}^{p} a_{1k}^*) + \sum_{j=1}^{q} (\sum_{k=1}^{p} a_{1k}^*) b_{1j}^* = (\sum_{k=1}^{p} a_{1k}^*) + (\sum_{k=1}^{p} a_{1k}^*)(\sum_{j=1}^{q} b_{1j}^*) = (\sum_{k=1}^{p} (A_1^*)_k) + (\sum_{k=1}^{p} (A_1^*)_k)(\sum_{j=1}^{q} (B_1^*)_j)$$

We get the accepted language of A times the accepted language of B.
Last but not least, we need to prove that there exists an automaton which gives us the kleene star of a rational language. Let $p, q \in \mathbb{N}$ and $A \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{p \times p}$ a transition matrix of a rational language and $P \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{p \times q}$, $Q \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle^{q \times p}$ like above, then

$$D := \begin{pmatrix} E_p & E_p \\ 0 & A + PQ \end{pmatrix}$$

Again, with the formula of the former proof, we get:

$$D^* = \begin{pmatrix} E_p & (A+PQ)^* \\ 0 & (A+PQ)^* \end{pmatrix}$$

The element in the upper right corner can be rewritten:

$$(A+PQ)^* = A^*(PQA^*)^*$$

The sum of the elements in the first row yields L^*. □

5 Conclusion

The former prove can be modified to also hold for formal power series in a semi-ring $\mathbb{S}\langle\langle\Sigma^*\rangle\rangle$ (ref. [Garding (1988)]). This theorem is called kleene-schützberg theorem (ref. [Garding (1988)]) and it makes a connection between formal power series and weighted automata. In summary, formal power series are really powerful, but also quite unintuitive at first. For futher reading my recommendation is the master thesis of Laura Wirth (ref. [Wirth (2022)]).

References

GARDING, TAMBOUR. 1988. *Algebra for computer science*. Sweden: Springer-Verlag New York Berlin Heidelberg London Paris Tokyo.

WIRTH. 2022. *Weighted automata, formal power series and weighted logic*. Konstanz, Germany: Springer Spektrum.