Bachelor thesis
Computing Science

Radboud University

Pumping Lemma for Deterministic Weighted Automata

Author:
Amber Pater
s1004019
`A.Pater@student.ru.nl`

Supervisor:
dr. J.C. Rot
`J.Rot@cs.ru.nl`

Second assessor:
prof. dr. J.H. Geuvers
`H.Geuvers@cs.ru.nl`

22nd March 2021

Abstract

This thesis is an introduction to the area of weighted automata; what they are, how they work and how they can be defined on the basis of semirings. Besides that, it goes into the determinism of weighted automata, which is not equivalent to non-determinism for this kind of automata. A pumping lemma that can be used to prove the non-determinism of a weighted language is given and applied to the language generating the Fibonacci sequence.

Contents

1 Introduction **2**

2 What are weighted automata? **4**
2.1 Simple example 4
2.2 Determining the output 5
2.3 Fibonacci 6

3 Different algebras **9**
3.1 Largest suffix 9
3.2 Shortest subword 10

4 Formal definition of weighted automata **12**
4.1 Semirings 12
4.2 Weighted automata as quintuple 13
4.3 Weighted languages 14

5 Pumping lemmas **15**
5.1 Classic pumping lemma 15
5.2 Pumping lemma for finite weighted automata 16

6 Application of the lemma **18**

7 Related work **22**

8 Conclusions **23**
8.1 Future work 23

Chapter 1

Introduction

Automata theory is a discipline in theoretical computing science that deals with the study of automata and formal languages, as well as with problems these automata can solve. This discipline is an important tool for computability and complexity theory, but it also finds practical application in the design of parsers, compliers and programming languages. An introduction to automata theory and formal languages, can be found in the paper of Hopcroft et al [5].

In regular automata theory we can use an automaton to check if a certain word is in a given language. There are multiple kinds of automata known in this area, but they all have the same property: they only have a binary check to show if words are accepted by a language or not.

To expand the use of automata theory, we can also look into weighted automata. Unlike the automata of regular automata theory, with weighted automata we do not only look at if a word is part of a certain language, but we also assign a specific weight to that word. Using weights is useful to check not only if a word can be read, but for example also in how many manners, what the shortest path will cost or to compute what the chances are for that word to be read.

Already in 1961, Schützenberger introduced weighted automata [9]. After that, multiple papers have been written on this topic. In 2009, the Handbook of weighted automata was published [2], which covered most of the findings of this research field until then.

We can see that this area has already been studied for quite some years. Unfortunately, most of the information that can be found on his topic is still on a highly technical, and specifically mathematical, level. Therefore, we will use this thesis to make this topic more accessible for an audience that does know about basic automata theory, but is not into the details of weighted automata yet.

Besides introducing weighted automata, we will also dive into some pumping lemmas. In 2020, Chattopadhyay et al published a paper introducing pumping lemmas for weighted automata [1]. We will expand on their work by formulating a new pumping lemma. For this lemma we will look into the determinism of weighted automata, something Mohri also worked on in 2009 [8]. Our lemma can be used to show the non-determinism of a given weighted language. This means that in weighted automata theory determinism and non-determinism are fundamentally different, in contradiction to regular automata theory.

To finish off, we will use our new pumping lemma to prove that the weighted language generating the Fibonacci sequence, which we will introduce when explaining what weighted automata are, can not be given by a deterministic weighted automaton.

Chapter 2

What are weighted automata?

To get a more intuitive understanding of weighted automata, we start with a few basic examples. Without going into the formal definitions of Chapter 4 yet, we examine how they work and discuss various ways on how to determine their output. After that, we will look at how using a different type of algebra changes their behaviour.

2.1 Simple example

The functioning of weighted automata is best shown through a basic example. Figure 2.1 shows a weighted automaton that takes a word and calculates the number of a's that the word ends with.

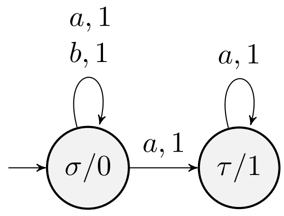

Figure 2.1: Simple weighted automaton calculating largest suffix of a's.

To show how the weighted automaton of Figure 2.1 processes a word, we consider the sample word '$abaaa$'. Like a regular automaton, we must find a path that matches this word. That is, we start in the initial state, in this case σ, which is indicated by the arrow coming in from outside. Then we follow an arrow with the letter 'a', then an arrow for 'b', and three more arrows for 'a'. In basic automata theory, we need to end in an accepting state, often indicated by a double circle. Weighted automata do not have such types of accepting states, instead all states have a certain weight. In this scenario, all states with a non-zero weight can be seen as accepting. For the example of Figure 2.1 that is only state τ.

To compute the output of a weighted automaton for a certain word, we must follow the following steps:

- Find all possible paths of that word.
- For every path: multiply the weights of the transitions with each other and multiply that with the weight of the state the path ends in.
- To get the total weight of a word, we sum up the weight of all its paths.

This way, we can compute the weight of our sample word '*abaaa*'.
There are four possible paths to read this word:

$$\begin{aligned}
&\sigma \to \sigma_a \to \sigma_{ab} \to \sigma_{aba} \to \sigma_{abaa} \to \sigma_{abaaa}: && (1 \cdot 1 \cdot 1 \cdot 1 \cdot 1) \cdot 0 = 0 \\
&\sigma \to \sigma_a \to \sigma_{ab} \to \sigma_{aba} \to \sigma_{abaa} \to \tau_{abaaa}: && (1 \cdot 1 \cdot 1 \cdot 1 \cdot 1) \cdot 1 = 1 \\
&\sigma \to \sigma_a \to \sigma_{ab} \to \sigma_{aba} \to \tau_{abaa} \to \tau_{abaaa}: && (1 \cdot 1 \cdot 1 \cdot 1 \cdot 1) \cdot 1 = 1 \\
&\sigma \to \sigma_a \to \sigma_{ab} \to \tau_{aba} \to \tau_{abaa} \to \tau_{abaaa}: && (1 \cdot 1 \cdot 1 \cdot 1 \cdot 1) \cdot 1 = 1
\end{aligned}$$

Summing up these weights, gives 3 as the total weight of the word '*abaaa*' which is indeed equal to its largest suffix of a's.

Checking that these four paths are indeed all the possible paths of our sample word '*abaaa*' is left as an exercise for the reader.

2.2 Determining the output

When working with weighted automata, it helps to be able to determine the output in multiple ways. After all, in some situations one method may be more suitable, whereas in other situations another technique could be more appropriate.

We have already seen one method to calculate the output: find all paths, determine their weights and add up the results. However, there is also another way.

In this method, we initially give every state a value:

- If the state is an initial state, it has value 1. (In theory other values are also possible, but this is not common.)

- If the state is not an initial state, it has value 0.

Then, when the weighted automaton processes a letter, all states are processed simultaneously. This way, we only have to look at one step in the path of the word at the time. How to update the states when reading a letter, is best explained through an example:

Let us use this method to calculate the weight of the word '*abaaa*' using the automata of Figure 2.1 again. First we take a look at the initial values of the states, in this case 1 for state σ and 0 for state τ. Then we look at how we can read the first letter, an a, from any state to any other state. When first reading an a there is only one way to end in state σ, namely the transition from that state itself. The update value of σ will be the previous value of state σ multiplied by the value of the path, thus $1 \cdot 1 = 1$. There is also just one way to end in state τ, namely with the path from σ. Therefore, the update value of τ after reading the first letter a will be: $1 \cdot 1 = 1$.

Next we need to read a b. By the same reasoning as before, the updated value of state σ is 1. Since we can not reach state τ when reading a letter b, the value of τ becomes 0.

The next letter a follows the same pattern as the first one. It becomes more interesting when reading the next letter a. Although we still have only one way to reach state σ, we can reach state τ in two ways: namely from σ and from τ. The update value of state τ becomes: the previous value of state σ multiplied by the value of the path, plus the previous value of state τ multiplied by the value of the other path, thus $1 \cdot 1 + 1 \cdot 1 = 2$.

To read the last letter we repeat this idea, but now these three paths lead to state τ. Finally, we need to multiply the values of the states after reading the last letter with their corresponding output value. For state σ this makes $1 \cdot 0 = 0$ and for state τ we get $3 \cdot 1 = 3$.

To keep an overview of the process we write it down as a table:

	σ	τ
Initial	1	0
a	1	1
b	1	0
a	1	1
a	1	2
a	1	3
Output	0	3

2.3 Fibonacci

To gain more insights into how weighted automata work, we consider another example.

We all know the famous Fibonacci sequence. Starting with $0, 1, 1, 2, 3, 5, ...$ and following the pattern in which each new number is the sum of the previous two.

In Figure 2.2 you can see a weighted automaton over the alphabet $\{a\}$. This automaton generates the language in which the word consisting of n a's (also written as: a^n) is mapped to the n-th Fibonacci number.

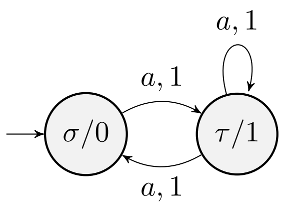

Figure 2.2: Weighted automaton generating the Fibonacci sequence

Theorem 1. *For all $n \in \mathbb{N}$, the weight of the word a^n is equal to the n-th Fibonacci number.*

Proof. To prove this theorem we use natural induction on n.

Base case: For $n = 0$, we get the empty word. This word has a weight of 0, since it directly is accepted in state σ.
For $n = 1$, we get the word 'a'. The only possible path from the initial state when reading 'a' is to state τ. Thus the weight of the word 'a' is $1 \cdot 1 = 1$.
Both these numbers are in line with the start of the Fibonacci sequence, thus the base cases hold.

Induction hypothesis: The weight of the word a^{n+2} is equal to the sum of the weights of the words a^{n+1} and a^n.

Induction step: We assume the IH holds for $n = k$ as well as for $n = k + 1$ for some $k \in \mathbb{N}$. Now we will prove the claim also holds for $n = k + 2$. Please note that since all weights are either zero or one, the weight of a path can also only be zero or one, by the rules of multiplication. To be more precise: all paths ending in state σ have a value of zero, all paths ending in state τ have a value of one. Therefore, σ can be seen as a non-accepting state.

There are three possibilities for the last transition (i.e. for reading the letter a for the $k + 2^{th}$ time):
(1) $\sigma \rightarrow \tau$
(2) $\tau \rightarrow \tau$
(3) $\tau \rightarrow \sigma$

The last possibility ends in a state with weight zero. Since multiplying by zero results in a total path weight of zero, these paths will not add anything to the total word weight, thus we can ignore these paths.

In the second option, we need to look at all paths of the word a^{k+1}, that end in state τ. Since σ can been seen as a non-accepting state, the total weight of the word a^{k+1} comes from paths ending in state τ. Thus, the weight of the paths of the word a^{k+2} taking (2) as its last transition, is equal to the weight of the word a^{k+1}.

Possibility (1) is all paths of the word a^{k+1} ending in state σ. Since σ can be seen as a non-accepting state, the weight of these paths is not included in the weight of the word a^{k+1}. If we look back an extra transition, we see that these paths have to be the paths of the word a^k ending in state τ. (Note that the only transition towards σ is coming from state τ). Fortunately, we know the weights of these paths, which is the weight of the word a^k.

Therefore, the weight of the word a^{k+2} is equal to the weight of the word a^{k+1} plus the weight of a^k.

By strong induction on k we can now conclude that, starting at the base cases, these weights must generate the Fibonacci sequence. $\square$

Chapter 3

Different algebras

Until now we have calculated the output of a word using the following method:

- For each path, we *multiply* all the weights of the transitions including the weight of the final state.
- For all paths together, *add up* all the results.

In short, we have used the operators 'addition' and 'multiplication'. Weighted automata are not restricted to these operators, they can also use other algebras, or so called "semirings". What those semirings are, precisely will be discussed in the next Chapter. We will first look at how they work, by considering a few other examples.

3.1 Largest suffix

One example is a weighted automaton that uses addition and the maximum function. Thus:

- For each path, we *add up* all the weights of the transitions including the weight of the final state.
- For all paths together, we take the *maximum* of all the results.

The automaton in Figure 3.1 uses that max-plus semiring, which is also known as the 'tropical semiring'. This automaton gives us again the largest suffix of a's of a word.

Let us check the word '*abaaa*' again: There are four possible paths to read this word:

$$\sigma \to \sigma_a \to \sigma_{ab} \to \sigma_{aba} \to \sigma_{abaa} \to \sigma_{abaaa} :$$
$$(0+0+0+0+0)+0 = 0$$
$$\sigma \to \sigma_a \to \sigma_{ab} \to \sigma_{aba} \to \sigma_{abaa} \to \tau_{abaaa} :$$
$$(0+0+0+0+0)+1 = 1$$
$$\sigma \to \sigma_a \to \sigma_{ab} \to \sigma_{aba} \to \tau_{abaa} \to \tau_{abaaa} :$$
$$(0+0+0+0+1)+1 = 2$$
$$\sigma \to \sigma_a \to \sigma_{ab} \to \tau_{aba} \to \tau_{abaa} \to \tau_{abaaa} :$$
$$(0+0+0+1+1)+1 = 3$$

Taking the maximum of these weights, gives 3 as the total weight of the word '*abaaa*'. This is indeed equal to its largest suffix of a's and is in line with the outcome of the previous automaton.

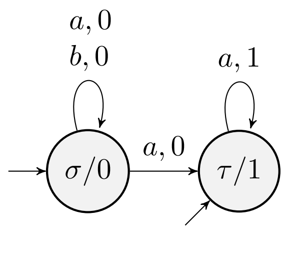

Figure 3.1: Max-plus weighted automaton returning the largest suffix of a's.

3.2 Shortest subword

In this next example, we use addition and minimum function to compute the shortest subword of b's in a word. (And if there is none, it outputs ∞.) Thus:

- For each path, we *add up* all the weights of the transitions including the weight of the final state.
- For all paths together, we take the *minimum* of all the results.

The automaton in Figure 3.2 uses this min-plus semiring. Let us check for the word '*babba*' what is its shortest subword consisting of only b's:

$$\rho \to \rho_b \to \rho_{ba} \to \rho_{bab} \to \rho_{babb} \to \rho_{babba}:$$
$$(0+0+0+0+0)+\infty=\infty$$
$$\rho \to \rho_b \to \rho_{ba} \to \rho_{bab} \to \rho_{babb} \to \sigma_{babba}:$$
$$(0+0+0+0+0)+\infty=\infty$$
$$\rho \to \rho_b \to \sigma_{ba} \to \tau_{bab} \to \tau_{babb} \to \upsilon_{babba}:$$
$$(0+0+1+1+0)+0=2$$
$$\sigma \to \tau_b \to \upsilon_{ba} \to \upsilon_{bab} \to \upsilon_{babb} \to \upsilon_{babba}:$$
$$(1+0+0+0+0)+0=1$$

Now we take the minimum of the results of all the paths and conclude that the shortest subword of 'b's consists indeed of only one 'b'.

To make this example complete, let's look at another word for this same automaton.

Take the word '$bbbabb$'. There are three possible paths to read this word:

$$\rho \to \rho_b \to \rho_{bb} \to \rho_{bbb} \to \rho_{bbba} \to \rho_{bbbab} \to \rho_{bbbab}:$$
$$(0+0+0+0+0+0)+\infty=\infty$$
$$\rho \to \rho_b \to \rho_{bb} \to \rho_{bbb} \to \sigma_{bbba} \to \tau_{bbbab} \to \tau_{bbbab}:$$
$$(0+0+0+0+1+1)+0=2$$
$$\sigma \to \tau_b \to \tau_{bb} \to \tau_{bbb} \to \upsilon_{bbba} \to \upsilon_{bbbab} \to \upsilon_{bbbab}:$$
$$(1+1+1+0+0+0)+0=3$$

The minimum of these outcomes is 2, which is indeed the shortest subword of b's, namely the last two letters of the word.

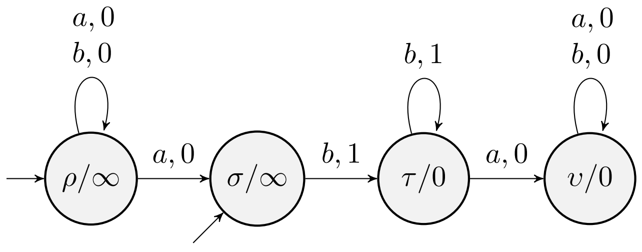

Figure 3.2: Weighted automaton computing shortest subword of b's.

Note that with some small changes, this automaton can easily be changed in one computing the longest subword of a certain letter.

Chapter 4

Formal definition of weighted automata

Now that we have a more intuitive understanding of how weighted automata work, we will look at how to properly define this kind of automata. We have seen in the previous Chapter that weighted automata can use different algebras, the so called semirings. Therefore to understand the definition of weighted automata, we first have to look at the definition of these semirings.

4.1 Semirings

A semiring is mathematical structure that is very similar to a ring. The only difference is that a semiring does not require each element to have an additive inverse.

Definition 1. *A semiring is an algebraic structure* $(S, \oplus, \otimes, \bar{0}, \bar{1})$ *with* S *a set, such that* $\bar{0}, \bar{1} \in S$ *and* $\oplus, \otimes : S \times S \to S$*, with the following properties:*

- $(S, \oplus)$ *is a commutative monoid with identity element* $\bar{0}$*, thus:*
  - $\forall a, b, c \in S : a \oplus (b \oplus c) = (a \oplus b) \oplus c$*;*
  - $\forall a \in S : \bar{0} \oplus a = a \oplus \bar{0} = a$*;*
  - $\forall a, b \in S : a \oplus b = b \oplus a$*.*
- $(S, \otimes)$ *is a monoid with identity element* $\bar{1}$*, thus:*
  - $\forall a, b, c \in S : (a \otimes b) \otimes c = a \otimes (b \otimes c)$*;*
  - $\forall a \in S : \bar{1} \otimes a = a \otimes \bar{1} = a$*.*
- *The operations (*$\oplus$ *and* $\otimes$*) are distributive, thus:*
  - $\forall a, b, c \in S : a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$*;*
  - $\forall a, b, c \in S : (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$*.*

- *Multiplication by $\bar{0}$ annihilates, thus:*
  - $\forall a \in S : \bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$.

A few examples of semirings are:

The natural semiring: $(\mathbb{N}_0, +, \cdot, 0, 1)$

The tropical semiring: $(\mathbb{R} \cup \infty, min, +, \infty, 0)$

The boolean semiring: $(\{0, 1\}, \vee, \wedge, 0, 1)$

To check that these structures are indeed semirings, is left as an exercise for the reader.

4.2 Weighted automata as quintuple

Similar to regular automata, weighted automata can be defined in terms of a tuple.

Definition 2. *Given a semiring S, any weighted automaton is described as a quintuple $(Q, \Sigma, \delta, i, o)$, with:*

Q *the finite set of states,*

Σ *the alphabet,*

$\delta : Q \to \Sigma \to (Q \to S)$ *the transition function,*

$i : Q \to S$ *the initial weight map,*

$o : Q \to S$ *the output weight map.*

For the set of states and the alphabet we do not need to go into more detail, as they are defined the same way in regular automata theory.

The transition function does differ from the one of regular automata. This function does not only tell for every state if a certain letter is read to which state it can go, but also what weight is assigned to that transition. These weights are elements of the chosen semiring.

The initial weight map and the output weight map return the initial, respectively the output, weight of a state. In our examples the initial weight is either $\bar{1}$ or $\bar{0}$, meaning it is either a starting state or it is not. However, automata can be defined with an initial weight equal to any element of the semiring, but we choose not to.

To illustrate this, we take a look back at our sample automaton in Figure 2.1. Given the semiring over $\mathbb{N}$, we can define this automaton as $(Q, \Sigma, \delta, i, o)$ with:

$Q = \{\sigma, \tau\}$;

$\Sigma = \{a, b\}$;

$\delta : Q \to \Sigma \to (Q \to S)$ with

$$\begin{aligned}
&\delta(\sigma)(a)(\sigma) = 1; \\
&\delta(\sigma)(a)(\tau) = 1; \\
&\delta(\sigma)(b)(\sigma) = 1; \\
&\delta(\tau)(a)(\tau) = 1.
\end{aligned}$$

$i : Q \to \mathbb{N}$ with

$$\begin{aligned}
&i(\sigma) = 1; \\
&i(\tau) = 0.
\end{aligned}$$

$o : Q \to \mathbb{N}$ with

$$\begin{aligned}
&i(\sigma) = 0; \\
&i(\tau) = 1.
\end{aligned}$$

4.3 Weighted languages

Last but not least, we also need to formally define the language of a weighted automata.

Definition 3. *Let de language of a given automaton be the function L that takes a word as input and returns the weight of that word, by the following:*

$$L : \Sigma^* \to S$$

$$L(w) = \bigoplus_{q \in Q} (i(q) \otimes \mathcal{L}(q)(w))$$

with

$$\mathcal{L} : Q \to \Sigma^* \to S$$

$$\mathcal{L}(q)(\lambda) = o(q)$$

$$\mathcal{L}(q)(aw) = \bigoplus_{p \in Q} (\delta(q)(a)(p) \otimes \mathcal{L}(p)(w)) \quad \forall a \in \Sigma$$

In this definition, we write Σ^* for the set of all possible words that can be made with the alphabet Σ.

Chapter 5

Pumping lemmas

In general automata theory, we can use the pumping lemma to prove that a certain language is not regular. Can we also use some kind of pumping lemma to say something about weighted automata? To answer this question we will first look at the pumping lemma for general automata theory. After that, we will define a version of the pumping lemma that is suitable for weighted automata.

5.1 Classic pumping lemma

Before diving into the pumping lemmas for weighted automata, let us first recall the classic pumping lemma for regular automata.

We know that for each regular language a finite automaton accepting that language exists. Thus a method to prove that a language is not regular, is to show that there does not exist any finite automaton accepting that language.

Lemma 1. *Let L be a regular language, which is recognized by a deterministic finite automaton M with k states. Let w be any word in L with $|w| \geq k$. Then w can be written as uvz, with $|uv| \leq k$ and $|v| > 0$, such that $\forall i \geq 0 : uv^iz \in L$.*

To illustrate the working of this lemma, we look at the language $L = \{a^nb^n | n \in \mathbb{N}\}$ over the alphabet $\{a, b\}$.

Assume L is regular, then a finite automaton M accepting L must exist. Let k be the number of states in M. Let $w = a^kb^k$ with $k \in \mathbb{N}$, then $|w| = 2k \geq k$. By the lemma, we can write $w = uvz$ where $|uv| \leq k$. This gives us $u = a^p$, $v = a^q$ and $z = a^rb^k$, with $p + q + r = k$ and $q \neq 0$. Now it must hold that $\forall i \geq 0 : uv^iz \in L$. In particular it must hold for $i = 2$, thus $uv^2z \in L$. But, $uv^2z = a^pa^{2q}a^rb^k = a^ka^qb^k \notin L$. This is in contradiction to the lemma. Therefore, our assumption was wrong, which makes L not regular.

5.2 Pumping lemma for finite weighted automata

In 2020, Chattopadhyay et al published a paper introducing some pumping lemmas for weighted automata [1]. Although they looked into ambiguity of weighted automata, they did not specify a lemma about the determinism of a given language.

The following new lemma can be used to prove the non-determinism of weighted automaton:

Lemma 2. *Assume $L : A^* \to S$ is a language recognized by a deterministic weighted finite automaton M, in which A is an alphabet and S is an arbitrary semiring. Then $\exists k \in \mathbb{N}$ such that $\forall w \in L, |w| \geq k : \exists u, v, z \in A^*$ and $\exists s \in S$ such that:*

- $w = uvz$
- $|uv| \leq k$ *and* $|v| > 0$
- $\forall i \in \mathbb{N} : L(uv^iz) = L(w) \cdot s^{i-1}$

Proof. We define N as the number of states in M. Take k as in the lemma and let $k = N$. Consider any given word $w \in L$. When this word is entered into the automaton M, various states will activate. Since the word w is at least of length N, the automaton will at some point come back to a state it has previously been in. Consider the first time that this happens and call this state O_n, conform Figure 5.1.

Define u as all the letters read prior to reaching state O_n, and v as all the letters read between the two occurrences of O_n (within the loop). Finally define z as the remaining letters, such that $w = uvz$. Note that $|v| > 0$, because the loop cannot consist of zero states.

Define the length of u as n and the length of v as m. Thus $|uv| = n+m$. This length varies based on the word w. Since k is defined as N, which is the largest possible value of $|uv|$ for all words w, this ensures that $|uv| \leq k$ for any word w.

Next, consider the value of $L(uv^iz)$ for any $i \in \mathbb{N}$. Let r_j be the weight of the j^{th} transition in the u part of the word. Let s_j and t_j be the j^{th} transition in the v, respectively z part, of the word. Then $L(uv^iz)$ is equal to: $r_1 \cdot r_2 \cdot \ldots \cdot r_n \cdot (s_1 \cdot s_2 \cdot \ldots \cdot s_m)^i \cdot t_1 \cdot t_2 \cdot \ldots \cdot t_l \cdot O_x$, where O_x represents the output of the final state of the word w, conform Figure 5.1.

We define:

$$r = r_1 \cdot r_2 \cdot \ldots \cdot r_n$$
$$s = s_1 \cdot s_2 \cdot \ldots \cdot s_m$$
$$t = t_1 \cdot t_2 \cdot \ldots \cdot t_l$$

Using this, we can write:

$$\begin{aligned} L(uv^iz) &= r_1 \cdot r_2 \cdot \ldots \cdot r_n \cdot (s_1 \cdot s_2 \cdot \ldots \cdot s_m)^i \cdot t_1 \cdot t_2 \cdot \ldots \cdot t_l \cdot O_x \\ &= r \cdot s^i \cdot t \cdot O_x \\ &= r \cdot s \cdot t \cdot O_x \cdot s^{i-1} \\ &= L(w) \cdot s^{i-1} \end{aligned}$$

This proves that $L(uv^iz) = L(w) \cdot s^{i-1}$ for any $i \in \mathbb{N}$, proving the theorem.

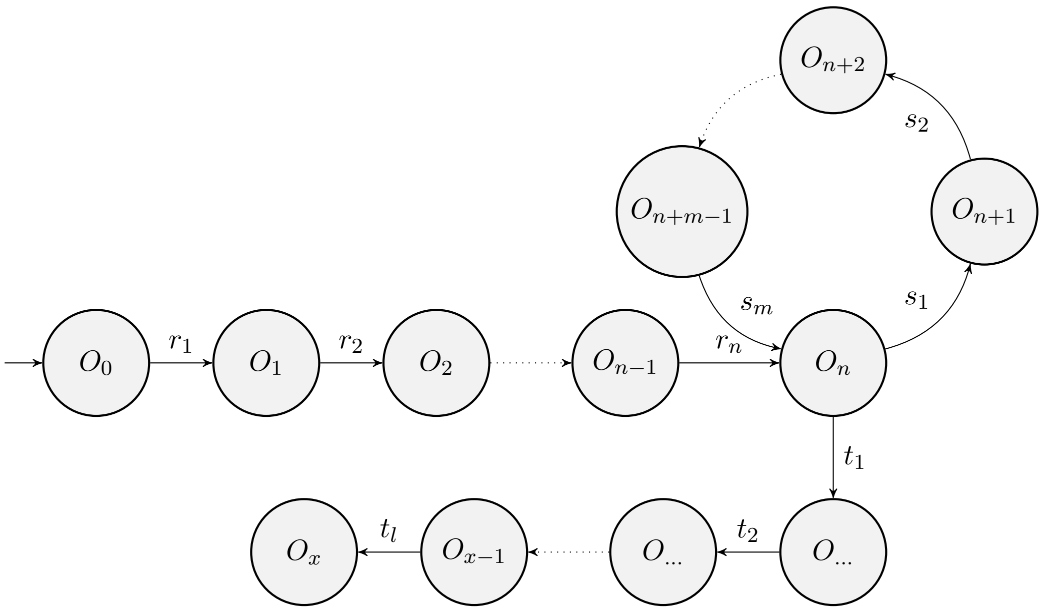

Figure 5.1: Deterministic weighted finite automaton M

□

It is good to note that, as opposed to regular automata theory, there is a difference between deterministic and non-deterministic weighted automata. Not all non-deterministic weighted automata can be transformed into a deterministic one, whereas this is possible for all non-deterministic regular automata. Therefore, not all weighted languages will meet Lemma 2. An example of such language will be shown in the next Chapter.

Chapter 6

Application of the lemma

We can now use this pumping lemma for weighted automata to prove that for the language that generates the Fibonacci sequence, which we introduced in section 2.3, no deterministic automaton can be given. But before we look into that proof, let us first look at the application of this lemma on a simpler language.

Consider the language L over the semiring $(\mathbb{N}, +, \cdot, 0, 1)$ with $L(a^n) = 2^n + 1$, as shown in Figure 6.1.

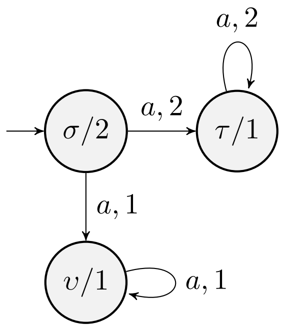

Figure 6.1: Automaton generating $L(a^n) = 2^n + 1$

Theorem 2. *The language L with $L(a^n) = 2^n + 1$ is not deterministic.*

Using the pumping lemma for weighted automata, we can now prove by contradiction that no deterministic automaton for this language L can be given.

Proof. Assume L is deterministic, then by Lemma 2 there exists a $k \in \mathbb{N}$ such that $\forall w \in L, |w| \geq k : \exists u, v, z \in \{a\}^*$ and $\exists s \in \mathbb{N}$ such that:

- $w = uvz$
- $|uv| \leq k$ and $|v| > 0$
- $\forall i \in \mathbb{N} : L(uv^i z) = L(w) \cdot s^{i-1}$

Let us take such k and consider the word $w = a^k$. Take u, v, z and s as mentioned in the lemma. We know that $a^k = uvz$, thus $uv^i z$ can be written as $a^{i \cdot m + l}$ with $m + l = k$ and $m > 0$.

Since it must hold that $\forall i \in \mathbb{N} : L(a^{i \cdot m+l}) = L(a^{m+l}) \cdot s^{i-1}$ we can conclude that:

$$\begin{aligned}
&L(a^{2m+l}) = L(a^{m+l}) \cdot s, \text{ thus} \\
&2^{2m+l} + 1 = (2^{m+l} + 1) \cdot s
\end{aligned}$$

and

$$\begin{aligned}
L(a^{3m+l}) &= L(a^{m+l}) \cdot s^2 \\
&= L(a^{2m+l}) \cdot s, \text{ thus} \\
2^{3m+l} + 1 &= (2^{2m+l} + 1) \cdot s
\end{aligned}$$

This gives us:

$$s = \frac{2^{2m+l} + 1}{2^{m+l} + 1} \text{ and } s = \frac{2^{3m+l} + 1}{2^{2m+l} + 1}$$

Therefore:

$$\begin{aligned}
&\frac{2^{2m+l} + 1}{2^{m+l} + 1} = \frac{2^{3m+l} + 1}{2^{2m+l} + 1} \\
&(2^{2m+l} + 1) \cdot (2^{2m+l} + 1) = (2^{3m+l} + 1) \cdot (2^{m+l} + 1) \\
&2^{4m+2l} + 2 \cdot 2^{2m+l} + 1 = 2^{4m+2l} + 2^{3m+l} + 2^{2m+l} + 1 \\
&2 \cdot 2^{2m+l} = 2^{3m+l} + 2^{2m+l}
\end{aligned}$$

Dividing both sides by 2^{m+l}, gives:

$$2 \cdot 2^m = 2^{2m} + 1$$

$\lightning$

This is a contradiction! For $m > 0$, the left sides results in an even number, since it is a multiple of two. The right hand side however is an power of two plus one and shall therefore be odd. A number cannot be both even and odd. Therefore, our assumption was wrong. Thus L is not deterministic. $\square$

A more interesting case is the language that generates the Fibonacci sequence, as introduced in Section 2.3. Because that language being deterministic means that for some staring index x and some interval i the x-th, $(x+i)$-th, $(x+2i)$-th, et cetera, number of the Fibonacci sequence are just multiples of each other, which is highly unlikely. Therefore we want to prove the theorem stating that for this language no deterministic automaton can be given.

Theorem 3. *The language generating the Fibonacci sequence is not deterministic.*

Proof. Let L be the language generating the Fibonacci sequence, as introduced in Section 2.3. Assume L is deterministic, then by Lemma 2 there exists a $k \in \mathbb{N}$ such that $\forall w \in L, |w| \geq k : \exists u, v, z \in \{a\}^*$ and $\exists s \in \mathbb{N}$ such that:

- $w = uvz$
- $|uv| \leq k$ and $|v| > 0$
- $\forall i \in \mathbb{N} : L(uv^iz) = L(w) \cdot s^{i-1}$

Let us take such k and consider the word $w = a^k$. Take u, v, z and s as mentioned in the lemma. We know that $a^k = uvz$ and since $|uv| \leq k$ and $|v| > 0$ we know that $v = a^j$ for some $j \in \mathbb{N}$ with $0 < j \leq k$.

By the lemma, it must hold that for all $i \in \mathbb{N} : L(uv^iz) = L(w) \cdot s^{i-1}$. Thus: $L(a^{k+(i-1)\cdot j}) = L(a^k) \cdot s^{i-1}$, or simplified: $L(a^{k+i\cdot j}) = L(a^k) \cdot s^i$.

The n-th number of the Fibonacci sequence can be found by the following formula (as proved by Horadam [6]):

$$\frac{\varphi^n - \psi^n}{\sqrt{5}} \text{ with } \varphi = \frac{1+\sqrt{5}}{2} \text{ and } \psi = \frac{1-\sqrt{5}}{2}$$

Therefore:

$$L(a^n) = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

Since it must hold that $\forall i \in \mathbb{N} : L(a^{k+i\cdot j}) = L(a^k) \cdot s^i$ we can conclude that:

$$L(a^{k+j}) = L(a^k) \cdot s, \text{ thus}$$

$$\frac{\varphi^{k+j} - \psi^{k+j}}{\sqrt{5}} = \frac{\varphi^k - \psi^k}{\sqrt{5}} \cdot s$$

and

$$\begin{aligned}
L(a^{k+2j}) &= L(a^k) \cdot s^2 \\
&= L(a^{k+j}) \cdot s, \text{ thus} \\
\frac{\varphi^{k+2j} - \psi^{k+2j}}{\sqrt{5}} &= \frac{\varphi^{k+j} - \psi^{k+j}}{\sqrt{5}} \cdot s
\end{aligned}$$

This gives us:

$$s = \frac{\varphi^{k+j} - \psi^{k+j}}{\sqrt{5}} \cdot \frac{\sqrt{5}}{\varphi^k - \psi^k} = \frac{\varphi^{k+j} - \psi^{k+j}}{\varphi^k - \psi^k}$$

and

$$s = \frac{\varphi^{k+2j} - \psi^{k+2j}}{\sqrt{5}} \cdot \frac{\sqrt{5}}{\varphi^{k+j} - \psi^{k+j}} = \frac{\varphi^{k+2j} - \psi^{k+2j}}{\varphi^{k+j} - \psi^{k+j}}$$

Therefore:

$$\begin{aligned}
\frac{\varphi^{k+j} - \psi^{k+j}}{\varphi^k - \psi^k} &= \frac{\varphi^{k+2j} - \psi^{k+2j}}{\varphi^{k+j} - \psi^{k+j}} \\
(\varphi^{k+j} - \psi^{k+j}) \cdot (\varphi^{k+j} - \psi^{k+j}) &= (\varphi^{k+2j} - \psi^{k+2j}) \cdot (\varphi^k - \psi^k) \\
\varphi^{2k+2j} - 2 \cdot \varphi^{k+j}\psi^{k+j} + \psi^{2k+2j} &= \varphi^{2k+2j} - \varphi^{k+2j}\psi^k - \varphi^k\psi^{k+2j} + \psi^{2k+2j} \\
-2 \cdot \varphi^{k+j}\psi^{k+j} &= -\varphi^{k+2j}\psi^k - \varphi^k\psi^{k+2j}
\end{aligned}$$

Dividing both sides by $-\varphi^k\psi^k$, gives:

$$\begin{aligned}
2 \cdot \varphi^j\psi^j &= \varphi^{2j} + \psi^{2j} \\
2 \cdot \varphi^j\psi^j &= (\varphi^j)^2 + (\psi^j)^2 \\
(\varphi^j)^2 - 2 \cdot \varphi^j\psi^j + (\psi^j)^2 &= 0 \\
(\varphi^j - \psi^j)^2 &= 0 \\
\varphi^j - \psi^j &= 0 \\
\varphi^j &= \psi^j
\end{aligned}$$

$\lightning$

This is a contradiction, since we know that $\varphi \neq \psi$ and $\varphi \neq -\psi$. Therefore, our assumption was wrong. Thus L is not deterministic. $\square$

Chapter 7

Related work

A lot of research has been done in the area of weighted automata. Back in 1961, Schützenberger published a paper on the definitions of the family of automata [9]. This paper was used as a basis for a lot of researchers. Most of the findings on this topic were put together in the Handbook of weighted automata by Droste et al in 2009 [2]. However, we will not go into further details on those related publications.

It is more interesting to look at related work about pumping lemmas and about determinism. Especially, the paper of Chattopadhyay on pumping lemmas for weighted automata from 2020 [1]. They have introduced multiple pumping lemmas for different properties of weighted automata. Although they did study ambiguity of weighted automata, where every word can only be read in at most one way, they did not specifically look into determinism. Mohri did study deterministic automata, but only to see their behaviour in certain algorithms [8].

Since to the best of our knowledge, no pumping lemma on deterministic weighted automata was published yet, we continued on the work of Chattopadhyay by coming up with a new pumping lemma that can be used to prove non-determinism of weighted automata.

Chapter 8

Conclusions

We have seen what weighted automata are, how they work and how they can be mathematically defined on the basis of semirings. Secondly, we introduced a new pumping lemma for weighted automata that can be used to prove the non-determinism of a weighted automaton. Therefore, we have also shown that determinism and non-determinism are not equivalent for weighted languages, as opposed to regular automata theory. Finally, we used our new lemma to show that for the language generating the Fibonacci sequence no deterministic automaton can be given.

8.1 Future work

For future work it would be interesting to look at a language generating the Fibonacci sequence over a complex semiring. Perhaps in that algebra it is possible to give a deterministic weighted automata for that language.

The knowledge we gained about determinism of weighted automata, can also be useful for the studies of self-learning algorithms on weighted automata. In the paper of van Heerdt et al [4] the language of Theorem 2 is given as an example for which their algorithm does not work. Perhaps any relation with determinism of weighted automata and the working of their algorithm can be found.

Bibliography

[1] Agnishom Chattopadhyay et al. “Pumping lemmas for weighted automata”. In: *CoRR* abs/2001.06272 (2020). arXiv: `2001.06272`. URL: `https://arxiv.org/abs/2001.06272`.

[2] Manfred Droste, Werner Kuich and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

[3] Manfred Droste and Dietrich Kuske. “Weighted automata”. In: *forthcoming handbook AutoMathA* (2012).

[4] Gerco van Heerdt et al. “Learning weighted automata over principal ideal domains”. In: *International Conference on Foundations of Software Science and Computation Structures*. Springer, Cham, 2020, 602–621.

[5] John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman. “Introduction to automata theory, languages, and computation”. In: *Acm Sigact News* 32.1 (2001), pp. 60–65.

[6] A.F. Horadam. “A generalized Fibonacci sequence”. In: *The American Mathematical Monthly* 68.5 (1961), pp. 455–459.

[7] H.W. Lenstra Jr. and F. Oort. “Groepentheorie”. Dictaat. 2014.

[8] Mehryar Mohri. “Weighted automata algorithms”. In: *Handbook of weighted automata*. Springer, 2009, pp. 213–254.

[9] Marcel Paul Schützenberger. “On the definition of a family of automata”. In: *Inf. Control.* 4.2-3 (1961), pp. 245–270.

[10] Alexandra Silva. “Languages and Automata, lecture notes”. Radboud University, 2016.

[11] Christian Wurm. “Gewichtete Automatenfibel”. Thesis. Heinrich Heine Universität Düsseldorf, 2014.