Bachelor's Thesis Computer Science

Minimising $\mathbb{Q}$-Weighted Finite Automata over $\mathbb{Z}$

Jasper Laumen

15th July 2024

First assessor:
Dr Jurriaan Rot

Second assessor:
Dr Sebastian Junges

Radboud University

Abstract

Weighted finite automata (WFAs) are a generalisation of nondeterministic finite automata, where transitions are additionally equipped with weights from a semiring. This induces a function from strings to weights. In this thesis, we provide an introduction to the theory of WFAs, and showcase one of the main results in Fliess' theorem. We prove a classical minimisation algorithm for WFAs over fields, and study an adaptation of this algorithm for minimisation over the integers as defined in [5]. Finally, we provide an implementation of this algorithm in Python, which allows for practical minimisation of WFAs over $\mathbb{Q}$ and $\mathbb{Z}$.

Contents

Chapter 1

Introduction

Nondeterministic finite automata (NFAs) are a fundamental concept in theoretical computer science and formal language theory. An NFA consists of a finite set of states, an input alphabet, a finite set of transitions, an initial state, and a set of accepting states. NFAs induce a function, called a *language*, which either accepts or rejects words.

Weighted finite automata (WFAs) are an extension of NFAs, where each transition is equipped with a weight. These automata induce a language function from strings to weights, where the weights are part of a semiring, such as the integers, rational numbers, or booleans. The mathematical theory behind these languages is the theory of *rational power series*. This topic has been thoroughly studied in the past [19, 4], but is still an active area of research [6]. In this thesis, we will approach these rational power series from a linear algebra and automata theoretic perspective.

Weighted finite automata have a wide variety of applications. One such applications is the modelling of neural networks [26]. Other applications include, but are not limited to, image processing [11], speech recognition [17], speech synthesis [16], bioinformatics [7, 21], and formal verification [1].

These applications motivate the ongoing research towards *minimising* WFAs. That is, finding a WFA that induces the same language, but uses a minimal amount of states. It is classical that a WFA with weights from a field can be minimised in polynomial time [12, 13]. However, the ring of integers $\mathbb{Z}$ are notably not a field, which means that this classical minimisation algorithm is not applicable to $\mathbb{Z}$. A recent paper [5] has provided an adaptation of this classical algorithm to minimise WFAs using weights in $\mathbb{Q}$ to a WFA using only integer weights, given that the induced language function produces only integer weights.

The main goal of this thesis is providing an overview of this algorithm and the theory needed to prove its correctness and time complexity, and give an implementation in Python. In Chapter 3, we will start with an intuitive definition of weighted finite automata, and use this to motivate the formal definition. We continue by looking at several examples of WFAs, and give two different methods to construct them, one of which is a fundamental result in the theory of WFAs.

In Chapter 4, we give a method to minimise WFAs over fields, and adapt this method to an algorithm that minimises WFAs over the integers. This algorithm can, given a WFA using weights in $\mathbb{Q}$, determine if the induced language is strictly integer valued, and if so, return an equivalent WFA using only integer weights, and otherwise return a counterexample in the form of a word, such that its weight is not an integer.

We finish Chapter 4 by looking at an applications of the minimisation algorithm, where we start with a WFA that uses non-integer weights, but induces an integer valued language, and end with an equivalent WFA with a minimal amount of states using only integer weights. Finally, we discuss an implementation of this algorithm in Python.

Chapter 2

Preliminaries

This chapter serves as a very quick overview of some definitions and results which do not directly relate to automata theory, but are used in this thesis. However, the upcoming sections are not suitable as an introduction to these topics, and only serve as a reference.

2.1 Ring and Field Theory

Weighted finite automata are defined over *semirings*, and we will often look at weighted finite automata over *fields*, which are a special case of semirings. In this section, we will introduce both of these concepts, starting with the *ring*. A ring is a mathematical structure where we can do addition and multiplication that follow specific properties.

Definition 2.1. A *ring* is a tuple $(R, +, \cdot, 0, 1)$, where R is an arbitrary set, and $+, \cdot$ are binary operators such that for each $a, b, c \in R$, it holds that

1. $(a + b) + c = a + (b + c)$;
2. $a + 0 = 0 + a = a$;
3. $a + b = b + a$;
4. There is $-a \in R$ such that $a + (-a) = 0$;
5. $(a \cdot b) \cdot c = a \cdot (b \cdot c)$;
6. $1 \cdot a = a \cdot 1 = a$;
7. $a(b + c) = ab + ac$;
8. $(a + b)c = ac + bc$.

Examples of rings are the integers $\mathbb{Z}$, the rational numbers $\mathbb{Q}$, and the set of $n \times n$ matrices over $\mathbb{R}$ for every $n \in \mathbb{N}_{>0}$.

A *semiring* is a ring without property 4. A *field* is a ring that also has the properties

9. $a \cdot b = b \cdot a$;
10. There is $a^{-1} \in R$ such that $a \cdot a^{-1} = 1$.

Examples of semirings include the natural numbers $\mathbb{N}$, and the boolean semiring $B = (\{0,1\}, \vee, \wedge, 0, 1)$. Examples of fields are the rational numbers $\mathbb{Q}$ and the real numbers $\mathbb{R}$. Notably, $\mathbb{Z}$ is not a field.

Definition 2.2. Let R be a ring. An element $a \in R$ is called a *zero divisor* if $a \neq 0$ and there exist $b, c \in R$ such that $ab = 0$ and $ca = 0$. A *domain* is a ring that also has property 9 (also called a *commutative* ring) and does not contain any zero divisors.

Definition 2.3. Let R be a ring. An *ideal* of R is a set $I \subseteq R$ such that for all $a, b \in I$ and $r \in R$ it holds that

1. $0 \in I$;
2. $a - b \in I$;
3. $ra \in I$;
4. $ar \in I$.

The ideal *generated* by some $a \in R$ is the set $\{ra \mid r \in R\}$. A domain D is called a *principal ideal domain* if every ideal I of D is generated by some $a \in D$. Examples of principle ideal domains are the integers $\mathbb{Z}$, fields such as $\mathbb{Q}$ and $\mathbb{R}$, and single variable polynomial rings over a field, such as $\mathbb{Q}[X]$.

2.2 Linear algebra

Since our formal definition of a WFA will use matrices, it is natural that the concept of a vector space will show up. However, vector spaces are only defined over fields, while we are often working in $\mathbb{Z}$, which is not a field, but it is a ring. In the case we are working with a ring, we will use *R-modules* instead of vector spaces. The definition of an R-module is completely analogous to the definition of a vector space, except for the set of scalars forming a ring R instead of a field F.

To differentiate between the vector space and the module generated by a set, we will use the notation $\langle v_1, v_2, \ldots, v_n \rangle_R$, with R a ring, to denote the R-module or vector space over R generated by the vectors $(v_1, v_2, \ldots, v_n)$. In particular, we will use $\langle v_1, v_2, \ldots, v_n \rangle_{\mathbb{Z}}$ and $\langle v_1, v_2, \ldots, v_n \rangle_{\mathbb{Q}}$ to denote the $\mathbb{Z}$-module and vector space generated by $(v_1, v_2, \ldots, v_n)$ over $\mathbb{Z}$ and $\mathbb{Q}$ respectively. Often, the ring R is omitted from the notation $\langle v_1, v_2, \ldots, v_n \rangle_R$ when it is clear from context which ring is used.

The next lemma will be used to find an integer basis of a module M when given a matrix whose columns span M.

Lemma 2.4 (**Smith, 1861 [23], Newman, 1997 [18]**). *Let R be a PID, and let $M = \langle v_1, v_2, \ldots, v_m \rangle$ be an R-module of rank r. Let $A \in R^{n \times m}$ be a matrix whose columns are $v_1, v_2, \ldots, v_m$. Then, the matrix A can be written in polynomial time as the product $S \cdot \widehat{A} \cdot T$, where $S \in R^{n \times n}$ and $T \in R^{m \times m}$ are invertible matrices, and $\widehat{A} = diag(d_1, d_2, \ldots, d_r, 0, \ldots, 0)$ such that $d_i \mid d_{i+1}$ for each $1 \leq i < r$.*

Let $D_i(A)$ be the greatest common divisor of the $i \times i$ minors of A. Then,

$$d_i = \frac{D_i(A)}{D_{i-1}(A)}. \tag{2.1}$$

Furthermore, the columns of the product $S \cdot \widehat{A}$ are an R-basis of M.

Lemma 2.5 (**Hadamard's inequality**). *Let A be a complex matrix whose columns are the vectors $v_1, v_2, \ldots, v_n$. Then,*

$$|det(A)| \leq \prod_{i=1}^{n} \|v_i\|.$$

Hadamard's inequality, along with Equation 2.1, will play an important role in proving the polynomial time complexity of the minimisation algorithm over the integers.

Definition 2.6. Let A be an $m \times n$ matrix, and let B be a $p \times q$ matrix. Then the *Kronecker product* of A and B, denoted $A \otimes B$, is the $pm \times qn$ block matrix given by

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}.$$

The Kronecker product follows the *Mixed product property*, meaning that $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$ for matrices A, B, C, D.

Definition 2.7. Let A and B be $m \times n$ matrices. Then the *Hadamard product* of A and B, denoted $A \odot B$, is the element wise product of A and B, given by

$$A \odot B = \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2n}b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \cdots & a_{mn}b_{mn} \end{pmatrix}.$$

Chapter 3

Weighted Finite Automata

In this chapter, we will provide an overview of the theory of weighted finite automata. We start with an intuitive definition, and use that to arrive at the formal definition. From there, we describe two different methods to construct new automata.

3.1 Definition

Deterministic finite automata and nondeterministic finite automata induce a language that is a function $\Sigma^* \to \{0,1\}$. We extend this notion of a language to a function $\Sigma^* \to R$, where R is an arbitrary semiring. Such a language is sometimes called a *weighted* language. Instead of deciding for each word "yes" (1) or "no" (0), a weighted language assigns a weight in R to each word in Σ^*. In this chapter, R will always be a semiring, unless stated otherwise.

Likewise, we extend the class of nondeterministic finite automata to the class of *weighted finite automata*. Informally, a weighted finite automaton (WFA) over R is a nondeterministic finite automaton (NFA), where each transition is equipped with a weight in R, and each state is equipped with an initial and final weight, also in R. A simple example of a WFA over $\mathbb{Q}$ is illustrated in Figure 3.1. The initial weights are indicated by arrows pointing towards a state, and the final weights are indicated by arrows leaving a state. The weights of every transition is shown alongside the corresponding letter.

The weight of a path $((q_0,q_1,\sigma_0),(q_1,q_2,\sigma_1),\ldots,(q_{n-1},q_n,\sigma_{n-1}))$, where each q_i is a state and each σ_i is a letter, is given by the initial weight of q_0, multiplied by the weight of each subsequent transition (q_i,q_{i+1},σ_i), and finally multiplied by the final weight of q_n. The weight of a word w is given by the sum of the weights of all paths accepting w. For example, the weight of the word aa in the automaton of Figure 3.1 is given by the weight of the two paths accepting aa, those being $((q_0,q_0,a),(q_0,q_1,a))$, and $((q_0,q_1,a),(q_1,q_1,a))$. The weights of these paths are calculated by $2\cdot 1\cdot\frac{1}{2}\cdot 1 = 1$ and $2\cdot\frac{1}{2}\cdot 1\cdot 1 = 1$ respectively, so the total weight of aa is $1+1=2$.

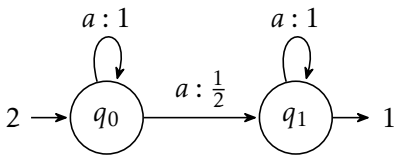

Figure 3.1: Simple WFA over $\mathbb{Q}$

In general, the weight of a word w in this automaton is $|w|$. To see this, notice that the weight of every individual path is 1. Furthermore, for a word w, there are $|w|$ paths accepting that word, because we can choose for which letter we can take the transition (q_0, q_1, a).

From this informal definition, it becomes clear why we require R to be a semiring. After all, there is no inherent ordering between different paths, so we would like addition to be commutative. However, this is not a requirement for multiplication, since there is indeed a fixed ordering for the transitions in a path.

Weighted languages accepted by a WFA are called *rational languages* [2]. By instantiating R to $(\{0,1\}, \vee, \wedge, 0, 1)$, also known as the Boolean semiring where addition is the logical operator "$\vee$" and multiplication is the logical operator "$\wedge$", we see that WFAs generalise NFAs, and thus every regular language is also a rational language. By instantiating R to $\mathbb{R}$, and further requiring that the weights of every transition from a state q_n sum to 1 for every n, we see that WFAs even generalise probabilistic automata, and thus every stochastic language is also a rational language.

Figure 3.2 illustrates a more complex example of a WFA that computes the perfect squares. However, verifying this computationally with our current definition is inefficient and cumbersome even for small words. Instead, we will reduce determining the weight of a word to matrix multiplication.

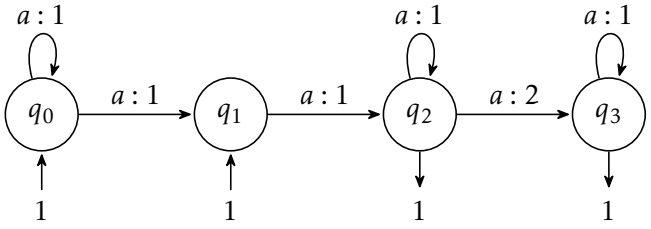

Figure 3.2: WFA over $\mathbb{N}$ computing the perfect squares

Definition 3.1. A *weighted finite automaton* over R is a tuple $(n, \Sigma, \alpha, \mu, \beta)$, where

- n is the amount of states;
- Σ is a finite alphabet;
- α is a row vector in R^n;
- μ is a map $\Sigma \to R^{n\times n}$;
- β is a column vector in R^n.

From our informal definition, α corresponds to the initial weights in the sense that α_i is the initial weight of the state q_i. Similarly, β_i is the final weight of the state q_i. Furthermore, μ contains the weight of every transition, such that $\mu(\sigma)_{ij}$ is the weight of the transition (q_i, q_j, σ). Notice that states do not have an explicit name in Definition 3.1. While these names are not strictly necessary, we give names to states in diagrammatic representations of WFAs for clarity purposes.

The weight of each word can now be computed relatively easily with matrix multiplication.

Definition 3.2. Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA over R. The *semantics* or *weights* of $\mathcal{A}$, denoted by $[\![\mathcal{A}]\!]$, is a language $\Sigma^* \to R$, such that $[\![\mathcal{A}]\!](w) = \alpha\mu(w)\beta$, where $\mu(w)$ is defined recursively by $\mu(\epsilon) = I_n$, where I_n is the $n \times n$ identity matrix, and $\mu(w\sigma) = \mu(w)\mu(\sigma)$.

Two WFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are said to be *equivalent* if $[\![\mathcal{A}_1]\!] = [\![\mathcal{A}_2]\!]$.

Example 3.3. The WFA of Figure 3.1 is the tuple $\mathcal{A} = (2, \{a\}, \alpha, \mu, \beta)$, where

$$\alpha = \begin{array}{c} \begin{matrix} q_0 & q_1 \end{matrix} \\ \begin{pmatrix} 2 & 0 \end{pmatrix} \end{array} \qquad \mu(a) = \begin{array}{c} \begin{matrix} & q_0 & q_1 \end{matrix} \\ \begin{matrix} q_0 \\ q_1 \end{matrix} \begin{pmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{pmatrix} \end{array} \qquad \beta = \begin{array}{c} \begin{matrix} \end{matrix} \\ \begin{matrix} q_0 \\ q_1 \end{matrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{array}.$$

Here, the matrix entries are labeled by the names of the corresponding states, where the row labels in the transition matrix represent the "from" state, and the columns labels represent the "to" state. Whenever there is no arrow, the weight is treated as 0. The weights of $\mathcal{A}$ are computed by

$$\begin{array}{lcll} [\![\mathcal{A}]\!](\epsilon) & = & \alpha\beta & = & 0 \\ [\![\mathcal{A}]\!](a) & = & \alpha\mu(a)\beta & = & 1 \\ [\![\mathcal{A}]\!](aa) & = & \alpha\mu(a)\mu(a)\beta & = & 2 \\ [\![\mathcal{A}]\!](aaa) & = & \alpha\mu(a)\mu(a)\mu(a)\beta & = & 3 \\ [\![\mathcal{A}]\!](\dots) & = & \dots & = & \dots \end{array}.$$

Example 3.4. The WFA of Figure 3.2 is the tuple $\mathcal{A} = (4, \{a\}, \alpha, \mu, \beta)$, where

$$\alpha = \begin{matrix} q_0 & q_1 & q_2 & q_3 \end{matrix} \begin{pmatrix} 1 & 1 & 0 & 0 \end{pmatrix} \qquad \mu(a) = \begin{matrix} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \end{matrix} \begin{matrix} \begin{matrix} q_0 & q_1 & q_2 & q_3 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix} \qquad \beta = \begin{matrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{matrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

Following this example, the state names will not be included as labels in the notation of a WFA anymore. We can compute the weights of $\mathcal{A}$ with

$$\begin{array}{lcll} [\![\mathcal{A}]\!](\epsilon) & = & \alpha\beta & = & 0 \\ [\![\mathcal{A}]\!](a) & = & \alpha\mu(a)\beta & = & 1 \\ [\![\mathcal{A}]\!](aa) & = & \alpha\mu(a)\mu(a)\beta & = & 4 \\ [\![\mathcal{A}]\!](aaa) & = & \alpha\mu(a)\mu(a)\mu(a)\beta & = & 9 \\ [\![\mathcal{A}]\!](\dots) & = & \dots & = & \dots \end{array}.$$

Of course, we can also prove that this automaton does indeed compute the perfect squares. First, notice that

$$\begin{pmatrix} 1 & 1 & x & y \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & x+1 & y+2x \end{pmatrix},$$

and by induction it follows that

$$\begin{aligned} \alpha\mu(a^n)\beta = \alpha\mu(a)^n\beta &= \begin{pmatrix} 1 & 1 & n & \sum_{i=0}^{n} 2i \end{pmatrix} \cdot \beta \\ &= n + \sum_{i=0}^{n} 2i \\ &= \sum_{i=0}^{n} (2i+1) \\ &= n^2. \end{aligned}$$

So far, we have only looked at WFAs with an alphabet consisting of just one letter. In the next example, we will look at a WFA with the alphabet $\{a, b\}$.

Example 3.5. The WFA in Figure 3.3 is given by the tuple $\mathcal{A} = (3, \{a, b\}, \alpha, \mu, \beta)$, where

$$\alpha = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \qquad \mu(a) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad \mu(b) = \begin{pmatrix} 1 & 1 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \beta = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}.$$

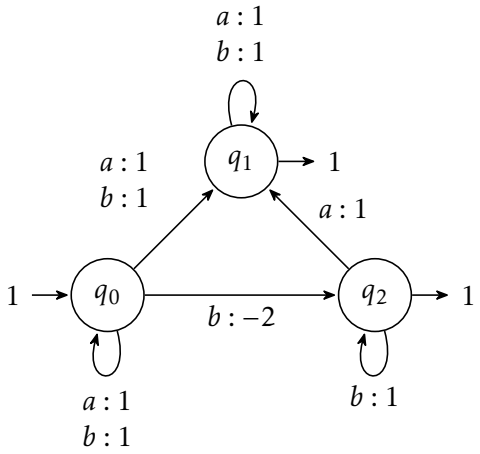

Figure 3.3: WFA over $\mathbb{Z}$ with multiple letters

This automaton computes $|w|_a - |w|_b$, and the weights are given by

$$\begin{array}{lcll} [\![\mathcal{A}]\!](\epsilon) & = & \alpha\beta & = & 0 \\ [\![\mathcal{A}]\!](a) & = & \alpha\mu(a)\beta & = & 1 \\ [\![\mathcal{A}]\!](b) & = & \alpha\mu(b)\beta & = & -1 \\ [\![\mathcal{A}]\!](aa) & = & \alpha\mu(a)\mu(a)\beta & = & 2 \\ [\![\mathcal{A}]\!](ab) & = & \alpha\mu(a)\mu(b)\beta & = & 0 \\ [\![\mathcal{A}]\!](ba) & = & \alpha\mu(b)\mu(a)\beta & = & 0 \\ [\![\mathcal{A}]\!](bb) & = & \alpha\mu(b)\mu(b)\beta & = & -2 \\ [\![\mathcal{A}]\!](\dots) & = & \dots & = & \dots \end{array}.$$

For simple rational languages, it is feasible to intuitively construct a corresponding WFA. However, this becomes more difficult for more complicated rational languages. In the next section, we will look at two techniques to construct WFAs when given its language.

3.2 Hankel Matrices

In Example 3.3, we looked at a WFA computing the perfect squares. Alternatively, we can define a product construction to construct an equivalent WFA as a product of the WFA in Figure 3.1 with itself. However, before we will look at this construction, we will first introduce the *Hankel matrix* of a WFA, named after German mathematician Hermann Hankel.

Definition 3.6. Let $f : \Sigma^* \to R$ be a language. The *Hankel matrix* of f, denoted H_f, is a matrix in $R^{\Sigma^* \times \Sigma^*}$ such that $H_f(w_1, w_2) = f(w_1 w_2)$ for each $w_1, w_2 \in \Sigma^*$, where $H_f(w_1, w_2)$ is the entry of H_f with row index w_1 and column index w_2. The Hankel matrix of a WFA $\mathcal{A}$ is the Hankel matrix of its semantics $[\![\mathcal{A}]\!]$. For notational simplicity, we write $H_\mathcal{A}$ instead of $H_{[\![\mathcal{A}]\!]}$.

We write $H_f(w_1, \cdot)$ and $H_f(\cdot, w_2)$ to denote the row and column vector indexed by w_1 and w_2 respectively.

Example 3.7. The Hankel matrix $H_\mathcal{A}$ of the automaton $\mathcal{A}$ in Figure 3.3 is given by the matrix

$$\begin{array}{c} \\ \epsilon \\ a \\ b \\ aa \\ ab \\ ba \\ bb \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccccccc} \epsilon & a & b & aa & ab & ba & bb & \ldots \end{array} \\ \begin{pmatrix} 0 & 1 & -1 & 2 & 0 & 0 & -2 & \cdots \\ 1 & 2 & 0 & 3 & 1 & 1 & -1 & \cdots \\ -1 & 0 & -2 & 1 & -1 & -1 & -3 & \cdots \\ 2 & 3 & 1 & 4 & 2 & 2 & 0 & \cdots \\ 0 & 1 & -1 & 2 & 0 & 0 & -2 & \cdots \\ 0 & 1 & -1 & 2 & 0 & 0 & -2 & \cdots \\ -2 & -1 & -3 & 0 & -2 & -2 & -4 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \end{array}.$$

In this finite subsection of the Hankel matrix, we can already see some structure. For example, the columns $H_\alpha(\cdot, ab)$ and $H_\alpha(\cdot, ba)$ are equal, and $H(\cdot, bb)$ can be written as $H(\cdot, \epsilon) - H_\alpha(\cdot, a) + H_\alpha(\cdot, b)$. In fact, this matrix has a finite rank, equal to 3. In Section 3.2.2, we will see that the Hankel matrix of a WFA always has a finite rank.

3.2.1 Hadamard Product

Considering the Hankel matrix of a WFA represents its semantics, we will require that the product of two WFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfies $H_{\mathcal{A}_1 \times \mathcal{A}_2} = H_{\mathcal{A}_1} \odot H_{\mathcal{A}_2}$, where $\odot$ is element wise matrix multiplication, also known as the *Hadamard product*, named after French mathematician Jacques Hadamard, or the *Schur product*, named after German mathematician Issai Schur. To find a suitable construction, it is useful to consider our first informal definition of a WFA. Namely, for every path of $\mathcal{A}_1$ and $\mathcal{A}_2$ accepting w, we want a path accepting w in $\mathcal{A}_1 \times \mathcal{A}_2$ with weight equal to the product of the weights of the two paths accepting w in $\mathcal{A}_1$ and $\mathcal{A}_2$. For every state q of $\mathcal{A}_1$ and every state r of

$\mathcal{A}_2$, we create a state (q,r) such that every transition $((q,r),(q',r'),\sigma)$ has the weight of the transition (q,q',σ) of $\mathcal{A}_1$ multiplied by the weight of the transition (r,r',σ) of $\mathcal{A}_2$. In matrix terms, this can be represented with the Kronecker product.

Definition 3.8. Let $\mathcal{A}_1 = (n,\Sigma,\alpha_1,\mu_1,\beta_1)$ and $(m,\Sigma,\alpha_2,\mu_2,\beta_2)$ be WFAs over R. The *Hadamard product* of $\mathcal{A}_1$ and $\mathcal{A}_2$, denoted $\mathcal{A}_1 \times \mathcal{A}_2$, is a WFA $\mathcal{B} = (n \cdot m, \Sigma, \alpha, \mu, \beta)$ over R, where

- $\alpha = \alpha_1 \otimes \alpha_2$;
- $\mu(\sigma) = \mu_1(\sigma) \otimes \mu_2(\sigma)$ for all $\sigma \in \Sigma$;
- $\beta = \beta_1 \otimes \beta_2$.

The idea behind this product aligns very closely with the regular product construction of two NFAs [10]. Indeed, the product of two WFAs over the Boolean semiring B corresponding to two NFAs is a WFA over B corresponding to the product of the two underlying NFAs.

Proposition 3.9. *Let $\mathcal{A}_1 = (n,\Sigma,\alpha_1,\mu_1,\beta_1)$ and $(\Sigma,m,\alpha_2,\mu_2,\beta_2)$ be WFAs over R, and let $\mathcal{A}_1 \times \mathcal{A}_2 = (n \cdot m, \Sigma, \alpha, \mu, \beta)$ be their product. Then, $H_{\mathcal{A}_1 \times \mathcal{A}_2} = H_{\mathcal{A}_1} \odot H_{\mathcal{A}_2}$, and in particular, $[\![\mathcal{A}_1 \times \mathcal{A}_2]\!] = [\![\mathcal{A}_1]\!] \cdot [\![\mathcal{A}_2]\!]$.*

Proof. We use structural induction on w to show that

$$(\alpha_1 \otimes \alpha_2)\mu(w) = (\alpha_1\mu_1(w)) \otimes (\alpha_2\mu_2(w)) \tag{3.1}$$

for all $w \in \Sigma^*$. For the base case $w = \epsilon$, we have

$$\begin{aligned}
(\alpha_1 \otimes \alpha_2)\mu(\epsilon) &= (\alpha_1 \otimes \alpha_2)I_{n\cdot m} \\
&= (\alpha_1 \otimes \alpha_2)(I_n \otimes I_m) \\
&= (\alpha_1 I_n) \otimes (\alpha_2 I_m) && \text{(mixed product property)} \\
&= \alpha_1 \otimes \alpha_2.
\end{aligned}$$

For the induction step, assume that $(\alpha_1 \otimes \alpha_2)\mu(w) = (\alpha_1\mu_1(w)) \otimes (\alpha_2\mu_2(w))$ for some $w \in \Sigma^*$ and let $\sigma \in \Sigma$. We have

$$\begin{aligned}
(\alpha_1 \otimes \alpha_2)\mu(w\sigma) &= (\alpha_1 \otimes \alpha_2)\mu(w)\mu(\sigma) \\
&= ((\alpha_1\mu_1(w)) \otimes (\alpha_2\mu_2(w)))\mu(\sigma) && \text{(IH)} \\
&= ((\alpha_1\mu_1(w)) \otimes (\alpha_2\mu_2(w)))(\mu_1(\sigma) \otimes \mu_2(\sigma)) \\
&= (\alpha_1\mu_1(w)\mu_1(\sigma)) \otimes (\alpha_2\mu_2(w)\mu_2(\sigma)) && \text{(mixed product property)} \\
&= (\alpha_1\mu_1(w\sigma)) \otimes (\alpha_2\mu_2(w\sigma)).
\end{aligned}$$

It now follows that for all $w \in \Sigma^*$,

$$\begin{aligned}
[\![\mathcal{A}_1 \times \mathcal{A}_2]\!](w) &= (\alpha_1 \otimes \alpha_2)\mu(w)(\beta_1 \otimes \beta_2) \\
&= ((\alpha_1 \mu_1(w)) \otimes (\alpha_2 \mu_2(w)))(\beta_1 \otimes \beta_2) && (3.1) \\
&= (\alpha_1 \mu_1(w)\beta_1) \otimes (\alpha_2 \mu_2(w)\beta_2) && \text{(mixed product property)} \\
&= [\![\mathcal{A}_1]\!](w) \cdot [\![\mathcal{A}_2]\!](w),
\end{aligned}$$

thus for all $w_1, w_2 \in \Sigma^*$,

$$\begin{aligned}
H_{\mathcal{A}_1 \times \mathcal{A}_2}(w_1, w_2) &= [\![\mathcal{A}_1 \times \mathcal{A}_2]\!](w_1 w_2) \\
&= [\![\mathcal{A}_1]\!](w_1 w_2) \cdot [\![\mathcal{A}_2]\!](w_1 w_2) \\
&= H_{\mathcal{A}_1}(w_1, w_2) \cdot H_{\mathcal{A}_2}(w_1, w_2),
\end{aligned}$$

which shows that element wise multiplication of $H_{\mathcal{A}_1}$ and $H_{\mathcal{A}_2}$ equals $H_{\mathcal{A}_1 \times \mathcal{A}_2}$. □

We are now ready to construct a different WFA representing the function $f(x) = x^2$.

Example 3.10. Let $\mathcal{A}$ be the WFA of Figure 3.1. The WFA $\mathcal{A} \times \mathcal{A}$ as illustrated in Figure 3.4 is the tuple $(4, \{a\}, \alpha, \mu, \beta)$, where

$$\alpha = \left(\begin{array}{cc|cc} 4 & 0 & 0 & 0 \end{array}\right) \qquad \mu(a) = \left(\begin{array}{cc|cc} 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{4} \\ 0 & 1 & 0 & \frac{1}{2} \\ \hline 0 & 0 & 1 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{array}\right) \qquad \beta = \left(\begin{array}{c} 0 \\ 0 \\ \hline 0 \\ 1 \end{array}\right).$$

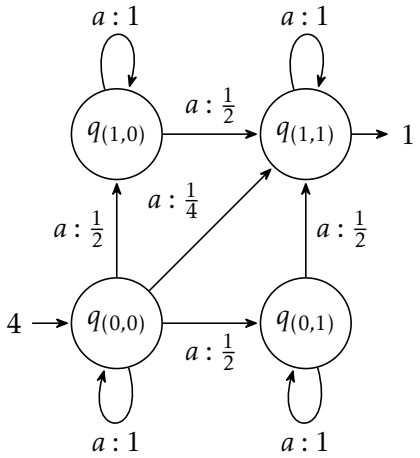

Figure 3.4: Product WFA

We have now seen two equivalent WFAs computing the perfect squares, both with 4 states. A natural question is whether there exists another equivalent WFA with fewer states. It turns out that this WFA does exist, making use of the linear relation $x^2 = 3(x-1)^2 - 3(x-2)^2 + (x-3)^2$, and we will construct this WFA in the upcoming section.

3.2.2 Fliess' Theorem

In this section, we discuss *Fliess' theorem*, which is one of the main results of the theory of weighted finite automata, and can be used to construct a WFA when given its Hankel matrix.

Theorem 3.11 (**Fliess, 1974 [8]**).

1. *Let $\mathcal{A}$ be a WFA with n states. Then, $rank(H_{\mathcal{A}}) \leq n$.*
2. *Let H_f be a Hankel matrix over a field F with rank n. Then there exists a WFA over F with n states such that $f = [\![\mathcal{A}]\!]$.*

Proof. We prove both parts of the theorem separately.

1. Assume that $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ is a WFA over R with n states. Let $\mathcal{A}_f \in R^{\Sigma^* \times \mathbb{Z}_n}$ be the matrix with rank at most n such that the i-th row is given by $\alpha\mu(i)$. Similarly, let $\mathcal{A}_b \in R^{\mathbb{Z}_n \times \Sigma^*}$ be the matrix with rank at most n such that the j-th column is given by $\mu(j)\beta$. We now have $\text{rank}(\mathcal{A}_f \cdot \mathcal{A}_b) \leq \min\Big(\text{rank}(\mathcal{A}_f), \text{rank}(\mathcal{A}_b)\Big) \leq n$, and $(\mathcal{A}_f \cdot \mathcal{A}_b)_{ij} = \alpha\mu(i)\mu(j)\beta = \alpha\mu(ij)\beta = (H_{\mathcal{A}})_{ij}$, so the i,j-th element of $(\mathcal{A}_f \cdot \mathcal{A}_b)$ is $H_{\mathcal{A}}(i,j)$, which means $H_{\mathcal{A}} = \mathcal{A}_f \cdot \mathcal{A}_b$ and thus $\text{rank}(H_{\mathcal{A}}) \leq n$.
2. **Intuitive idea.** We (non-constructively) find a basis for all columns of the Hankel matrix, and then construct a WFA in terms of this basis.

   Construction. Assume that H is a Hankel matrix with rank n. Then there exists a basis $(H(\cdot, v_1), H(\cdot, v_2), ..., H(\cdot, v_n))$ for all columns of H. Define the initial vector α such that $\alpha_j = H(\epsilon, v_j)$.

   By definition, there exist $B_1, B_2, ..., B_n$ such that $H(\cdot, \epsilon) = \sum_{i=1}^{n} B_i H(\cdot, v_i)$. Define the final vector β such that $\beta_i = B_i$.

   For all $\sigma \in \Sigma$ and v_j, we can write $H(\cdot, \sigma v_j)$ in terms of the basis such that $H(\cdot, \sigma v_j) = \sum_{i=1}^{n} \gamma_{ij}^{\sigma} H(\cdot, v_i)$ for some $\gamma_{1j}^{\sigma}, \gamma_{2j}^{\sigma}, ..., \gamma_{nj}^{\sigma}$. For each $\sigma \in \Sigma$, let $M(\sigma)$ be the matrix such that $M(\sigma)_{ij} = \gamma_{ij}^{\sigma}$. Define $M(w)$ for $w \in \Sigma^*$ recursively by $M(\epsilon) = I_n$ and $M(w\sigma) = M(w)M(\sigma)$. We define the transition matrices μ such that $\mu(\sigma)_{ij} = M(\sigma)_{ij}$.

   These choices for the initial vector, final vector, and transition matrices may seem arbitrary, but they do have an interpretation. We see each state as one of the basis elements. The initial vector can be seen as "initialising" these states to the empty word. The transition matrices then ask, given we have written a specific word w in terms of the basis, how do we write $w\sigma$ in terms of this basis? Finally, the final

vector then applies the "starting" coefficients to the basis elements, namely, how to give the weight of the empty word in terms of the basis.

Correctness. Let $w \in \Sigma^*$ arbitrary, and P be the row matrix in F^{Σ^*} such that P is 1 at index w and 0 elsewhere. Then

$$\begin{aligned} f(w) &= H(w, \epsilon) \\ &= PH(\cdot, \epsilon) \\ &= P\left(\sum_{i=1}^{n} B_i H(\cdot, v_i)\right) \\ &= \sum_{i=1}^{n} PB_i H(\cdot, v_i) \\ &= \sum_{i=1}^{n} B_i PH(\cdot, v_i) \\ &= \sum_{i=1}^{n} B_i H(w, v_i), \end{aligned} \tag{3.2}$$

where we used the left distributivity of matrix multiplication in the third to last step.

We will use structural induction on $w = \sigma_1, \sigma_2, ..., \sigma_k \in \Sigma^*$ to show that $H(\cdot, wv_j) = \sum_{i=1}^{n} M(w)_{ij} H(\cdot, v_i)$. The base case $w = \epsilon$ follows from the fact that $(I_n)ij = 1$ if and only if $i = j$. The base case $w = \epsilon \in \Sigma$ follows by definition of $M(\sigma)$. For the recursive step, assume that equality holds for $w_1, w_2 \in \Sigma^*$. Let P again be the row matrix in F^{Σ^*} such that P is 1 at index w_1 and 0 elsewhere. Then for $w = w_1 w_2$, we have

$$\begin{aligned} H(\cdot, wv_j) &= H(w_1, w_2 v_i) \\ &= PH(\cdot, w_2 v_i) \\ &= P\left(\sum_{i=1}^{n} M(w_2)_{ij} H(\cdot, v_i)\right) && \text{(IH)} \\ &= \sum_{i=1}^{n} PM(w_2)_{ij} H(\cdot, v_i) && \text{(distributivity)} \\ &= \sum_{i=1}^{n} M(w_2)_{ij} PH(\cdot, v_i) \\ &= \sum_{i=1}^{n} M(w_2)_{ij} H(w_1, v_i) \\ &= \sum_{i=1}^{n} M(w_2)_{ij} H(\cdot, w_1 v_i) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{n} \left(M(w_2)_{ij} \sum_{k=1}^{n} M(w_1)_{ki} H(v_k) \right) && \text{(IH)} \\
&= \sum_{i=1}^{n} \left(\sum_{k=1}^{n} M(w_2)_{ij} M(w_1)_{ki} H(v_k) \right) \\
&= \sum_{k=1}^{n} \left(\sum_{i=1}^{n} M(w_2)_{ij} M(w_1)_{ki} H(v_k) \right) && \text{(commutativity of sum)} \\
&= \sum_{k=1}^{n} \left(\sum_{i=1}^{n} M(w_1)_{ki} M(w_2)_{ij} H(v_k) \right) && \text{(commutativity of product)} \\
&= \sum_{k=1}^{n} (M(w_1)M(w_2))_{kj} H(v_k) \\
&= \sum_{k=1}^{n} M(w_1 w_2)_{kj} H(v_k) \\
&= \sum_{k=1}^{n} M(w)_{kj} H(v_k).
\end{aligned}
$$

Now, let $\mathcal{A}$ be the automaton $(n, \Sigma, \alpha, \mu, \beta)$. Then

$$
\begin{aligned}
f(w) &= \sum_{i=1}^{n} \beta_i H(\epsilon, w v_i) && (3.2) \\
&= \sum_{i=1}^{n} \left(\beta_i \sum_{j=1}^{n} \mu(w)_{ji} H(\epsilon, v_j) \right) && \text{(induction)} \\
&= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \beta_i \mu(w)_{ji} H(\epsilon, v_j) \right) && \text{(distributivity)} \\
&= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} H(\epsilon, v_j) \mu(w)_{ji} \beta_i \right) && \text{(commutativity of product)} \\
&= \alpha \mu(w) \beta && \text{(definition of matrix multiplication)} \\
&= [\![\mathcal{A}]\!](w),
\end{aligned}
$$

Thus there exists an automaton $\mathcal{A}$ with n states with $f = [\![\mathcal{A}]\!]$.

This completes the proof of both parts of the theorem. □

Example 3.12. The Hankel matrix H_f of the language $f(w) \to |w|_a^2$ is given by

$$\begin{array}{c} \\ \epsilon \\ a \\ aa \\ aaa \\ aaaa \\ aaaaa \\ \dots \end{array} \begin{array}{c} \begin{array}{ccccccc} \epsilon & a & aa & aaa & aaaa & aaaaa & \dots \end{array} \\ \begin{pmatrix} 0 & 1 & 4 & 9 & 16 & 25 & \cdots \\ 1 & 4 & 9 & 16 & 25 & 36 & \cdots \\ 4 & 9 & 16 & 25 & 36 & 49 & \cdots \\ 9 & 16 & 25 & 36 & 49 & 64 & \cdots \\ 16 & 25 & 36 & 49 & 64 & 81 & \cdots \\ 25 & 36 & 49 & 64 & 81 & 100 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \end{array}.$$

Notice that $H_f(\cdot, w) - 3H_f(\cdot, wa) + 3H_f(\cdot, waa) = H_f(\cdot, waaa)$ for all $w \in \{a\}^*$. Since the vectors $H_f(\cdot, \epsilon)$, $H_f(\cdot, a)$ and $H_f(\cdot, aa)$ are linearly independent, it follows that these vectors form a basis for the columns of H_f.

Following the construction from Theorem 3.11 we have to write $H(\cdot, \epsilon)$ in terms of the basis. We easily find $H(\cdot, \epsilon) = 1 \cdot H(\cdot, \epsilon) + 0 \cdot H(\cdot, a) + 0 \cdot H(\cdot, aa)$. This gives us the final vector

$$\beta = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

For the transition matrix $\mu(a)$, we have to write $H(\cdot, a)$, $H(\cdot, aa)$ and $H(\cdot, aaa)$ in terms of the basis. In this case we find

$$\begin{aligned} H(\cdot, a) &= 0 \cdot H(\cdot, \epsilon) + 1 \cdot H(\cdot, a) + 0 \cdot H(\cdot, aa) \\ H(\cdot, aa) &= 0 \cdot H(\cdot, \epsilon) + 0 \cdot H(\cdot, a) + 1 \cdot H(\cdot, aa) \\ H(\cdot, aaa) &= 1 \cdot H(\cdot, \epsilon) - 3 \cdot H(\cdot, a) + 3 \cdot H(\cdot, aa), \end{aligned}$$

giving the transition matrix

$$\mu(a) = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -3 \\ 0 & 1 & 3 \end{pmatrix}.$$

Finally, the initial vector is given by

$$\alpha = \begin{pmatrix} H_f(\epsilon, \epsilon) & H_f(\epsilon, a) & H_f(\epsilon, aa) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 4 \end{pmatrix}.$$

The resulting WFA, computing the perfect squares, is illustrated in Figure 3.5.

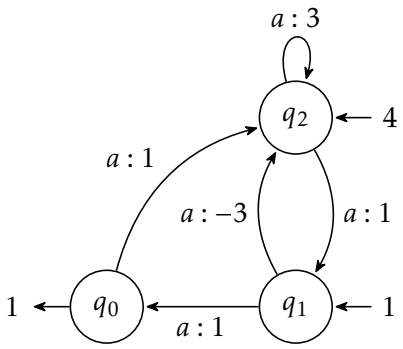

Figure 3.5: WFA computing the perfect squares with 3 states

The choice of basis has a direct influence on the resulting WFA. In Example 3.12, we could have chosen any three columns of H_f as a basis. For example, consider the basis $(H(\cdot,a), H(\cdot,aaa), H(\cdot,aaaaa))$. Simple linear algebra gives us

$$\begin{aligned}
H(\cdot,\epsilon) &= \tfrac{15}{8}\cdot H(\cdot,a) - \tfrac{5}{4}H(\cdot,aaa) + \tfrac{3}{8}H(\cdot,aaaaa)\\
H(\cdot,aa) &= \tfrac{3}{8}\cdot H(\cdot,a) + \tfrac{3}{4}H(\cdot,aaa) - \tfrac{1}{8}H(\cdot,aaaaa)\\
H(\cdot,aaaa) &= -\tfrac{1}{8}\cdot H(\cdot,a) + \tfrac{3}{4}H(\cdot,aaa) + \tfrac{3}{8}H(\cdot,aaaaa)\\
H(\cdot,aaaaaa) &= \tfrac{3}{8}\cdot H(\cdot,a) - \tfrac{5}{4}H(\cdot,aaa) + \tfrac{15}{8}H(\cdot,aaaaa).
\end{aligned}$$

This gives another WFA with

$$\alpha = \begin{pmatrix}1 & 9 & 25\end{pmatrix} \qquad \mu(a) = \begin{pmatrix}\frac{3}{8} & -\frac{1}{8} & \frac{3}{8}\\ \frac{3}{4} & \frac{3}{4} & -\frac{5}{4}\\ -\frac{1}{8} & \frac{3}{8} & \frac{15}{8}\end{pmatrix} \qquad \beta = \begin{pmatrix}\frac{15}{8}\\ -\frac{5}{4}\\ \frac{3}{8}\end{pmatrix}.$$

This WFA is clearly a lot more complicated, but still equivalent to the WFA in Figure 3.5. While Fliess' theorem can be used to find so called *minimal* automata, it definitely does not guarantee that the end result is in its simplest form. However, it is possible to find a minimal equivalent WFA using only integer weights for any WFA whose semantics are integer valued, but that requires a more involved algorithm, and will be the main focus of the next chapter. Unfortunately, the same does not hold for semantics over the natural numbers. In general, many decision problems for WFAs over arbitrary semirings are undecidable [14].

Chapter 4

Minimisation

In this chapter, we will build up to an algorithm that, given a WFA $\mathcal{A}$, determines if $[\![\mathcal{A}]\!]$ is integer valued, and if it is, returns a minimal WFA equivalent to $\mathcal{A}$ using only integer weights, and otherwise returns a word w such that $[\![\mathcal{A}]\!](w) \notin \mathbb{Z}$. This algorithm, first published in [5], is a variation of the classical minimisation algorithm for WFAs over fields [22, 4]. We will first introduce this classical algorithm, and then describe how it can be adapted to minimise WFAs over the integers. Finally, we will briefly discuss an implementation of the minimisation algorithms in Python.

4.1 Minimisation over fields

In this section, a WFA is always defined over a field F, unless stated otherwise.

The core ideas of minimisation are closely related to Fliess' theorem, where we saw that the amount of states in a WFA is bound by the rank of its Hankel matrix. In the proof of Theorem 3.11, we bounded the rank of the Hankel matrix by the rank of the matrices $\mathcal{A}_f$ and $\mathcal{A}_b$. These matrices span the *forward space* and *backward space* of a WFA.

Definition 4.1. Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA. The *forward space* of $\mathcal{A}$, denoted $\mathcal{F}_\mathcal{A}$, is the vector space $\langle \alpha\mu(w) \mid w \in \Sigma^* \rangle$. The *backward space* of $\mathcal{A}$, denoted $\mathcal{B}_\mathcal{A}$, is the vector space $\langle \mu(w)\beta \mid w \in \Sigma^* \rangle$.

To find a WFA with states equal to the rank of its Hankel matrix, we will first construct a WFA with n equal to the dimension of the forward space, called the *forward conjugate*, and then construct a WFA with n equal to the dimension of the backward space of this forward conjugate, called the *backward conjugate*. In order to construct these conjugates, we need to find a basis of the forward and backward space, which we will use to write α, μ and β in terms of this basis.

Polynomial time algorithms that find these bases are stated in Figure 4.1. They date back to [24], but the given version is from [5].

```
def compute_forward_basis(n, Σ, α, μ, β):
    F = {α}
    while there is (f, σ) ∈ F × Σ such that f μ(σ) ∉ ⟨F⟩:
        F = F ∪ {f μ(σ)}
    return F

def compute_backward_basis(n, Σ, α, μ, β):
    B = {β}
    while there is (b, σ) ∈ B × Σ such that μ(σ)b ∉ ⟨B⟩:
        B = B ∪ {μ(σ)b}
    return B
```

Figure 4.1: Polynomial time algorithms to compute a forward basis and backward basis of a WFA [5]

Lemma 4.2. *The algorithm* `compute_forward_basis` *in Figure 4.1 computes a basis of the forward space of a WFA in* $\mathcal{O}(n^3|\Sigma|)$ *time. Likewise, the algorithm* `compute_backward_basis` *computes a basis of the backward space in* $\mathcal{O}(n^3|\Sigma|)$.

Proof. Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA. Since $\{\alpha\mu(w) \mid w \in \Sigma^*\}$ is the smallest set containing a that is closed under postmultiplication with $\mu(\sigma)$ for all $\sigma \in \Sigma$, its span $\mathcal{F}_\mathcal{A}$ is the smallest vector space with this property. When `compute_forward_basis` returns F, by the loop condition, it holds that all elements of F are closed under postmultiplication with $\mu(\sigma)$, and by distributivity of matrix multiplication, all elements of $\langle F\rangle$ are also closed under postmultiplication with $\mu(\sigma)$. The minimality of $\langle F\rangle$ follows from the fact that its dimension increases by one each iteration of the loop. Thus, F is the basis of the forward space of $\mathcal{A}$.

Similarly, $\langle \mu(w)\beta \mid w \in \Sigma^*\rangle$ is the smallest vector space that is closed under premultiplication with $\mu(\sigma)$ for all $\sigma \in \Sigma$. When `compute_backward_basis` returns B, by the loop condition, all elements of B are closed under premultiplication with $\mu(\sigma)$, and from distributivity of matrix multiplication, $\langle B\rangle$ is also closed under premultiplication with $\mu(\sigma)$. The minimality of $\langle B\rangle$ again follows from the fact that its dimension increases by one each loop iteration. Thus, B is the basis of the backward space of $\mathcal{A}$.

Since each element of F and B has n entries, $\langle F\rangle$ and $\langle B\rangle$ can have dimensions of at most n. Each iteration of the `while` loop, the dimensions of $\langle F\rangle$ and $\langle B\rangle$ increase by one. Thus, the `while` loop can iterate at most $\mathcal{O}(n)$ times. Note that in each iteration, we only have to check the condition of the `while` loop for words we have not checked before. For each word, there are $|\Sigma|$ pairs to check. Thus, there are $\mathcal{O}(n|\Sigma|)$ pairs $(f, \sigma) \in F \times \Sigma$ and $(\sigma, b) \in \Sigma \times B$ to check in total. Each check is equivalent to checking whether the dimension of $\langle F\rangle$ or $\langle B\rangle$ is equal to the dimension of $\langle F \cup \{f\mu(\sigma)\}\rangle$ and $\langle B \cup \{\mu(\sigma)b\}\rangle$ respectively. This can be done by combining every vector in a matrix and using Gaussian elimination.

Traditionally, this takes $\mathcal{O}(n^3)$ time, but we do not need to reduce the entire matrix every time. We can store the reduced matrix after each iteration, and add the new vector to this reduced matrix. Then, only this new vector needs to be reduced, lowering the complexity to $\mathcal{O}(n^2)$. In summary, we have to check $\mathcal{O}(n|\Sigma|)$ pairs, and each check takes $\mathcal{O}(n^2)$ time, resulting in a $\mathcal{O}(n^3|\Sigma|)$ time algorithm. □

We will use these bases to construct new WFAs. The basis of the forward space will be used to construct the *forward conjugate*, which encodes the information of the initial vector α in μ and β. Likewise, the basis of the backward space will be used to construct the *backward* conjugate, which encodes the information of the final vector β in α and μ.

Definition 4.3. Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA, and let F be a matrix whose rows form a basis of $\mathcal{F}_\mathcal{A}$. The *forward conjugate* of $\mathcal{A}$, denoted $\overrightarrow{\mathcal{A}}$, is the WFA $(\overrightarrow{n}, \Sigma, \overrightarrow{\alpha}, \overrightarrow{\mu}, \overrightarrow{\beta})$, where

- $\overrightarrow{\alpha} F = \alpha$;
- $\overrightarrow{\mu}(\sigma) F = F\mu(\sigma)$;
- $\overrightarrow{\beta} = F\beta$.

From the fact that the rows of F form a basis of $\mathcal{F}_\mathcal{A}$, which contains α, it follows that $\overrightarrow{\alpha}$ is well defined, and also unique. Furthermore, the rows of $F\mu(\sigma)$ are vectors in $\mathcal{F}_\mathcal{A}$, and thus $\overrightarrow{\mu}(\sigma)$ is well defined and unique for all $\sigma \in \Sigma$.

When F is formed using the algorithm in Figure 4.1, something interesting happens. Since we start with $W = \{\epsilon\}$, we have that $\alpha\mu(\epsilon) = \alpha$ is in the basis. From $\overrightarrow{\alpha} F = \alpha$, it then follows that $\overrightarrow{\alpha}$ is a vector being equal to 0 at every entry, except for one entry where it is equal to 1. This essentially trivialises the initial vector.

The definition for the backward conjugate is analogous, but mirrored.

Definition 4.4. Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA, and let B be a matrix whose rows form a basis of $\mathcal{B}_\mathcal{A}$. The *backward conjugate* of $\mathcal{A}$, denoted $\overleftarrow{\mathcal{A}}$, is the WFA $(\overleftarrow{n}, \Sigma, \overleftarrow{\alpha}, \overleftarrow{\mu}, \overleftarrow{\beta})$, where

- $\overleftarrow{\alpha} = \alpha B$;
- $B\overleftarrow{\mu}(\sigma) = \mu(\sigma)B$;
- $B\overleftarrow{\beta} = \beta$.

For the backward conjugate, we now have that β will be trivialised when B has been found using the algorithm in Figure 4.1.

Of course, these constructions would not make sense if they change the semantics of the WFA. However, it turns out that the forward and backward conjugate are indeed equivalent to the original WFA. To prove this, we will first use induction to show that $\overrightarrow{\mu}(w)F = F\mu(w)$ and $B\overleftarrow{\mu}(w) = \mu(w)B$ for all $w \in \Sigma^*$.

Lemma 4.5. *Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA, and let F be a matrix whose rows form a basis of $\mathcal{F}_\mathcal{A}$. If $\overrightarrow{\mathcal{A}} = (\overrightarrow{n}, \Sigma, \overrightarrow{\alpha}, \overrightarrow{\mu}, \overrightarrow{\beta})$ is constructed using F, then $\overrightarrow{\mu}(w)F = F\mu(w)$ for all $w \in \Sigma^*$.*

Proof. We will use induction on the length of $w \in \Sigma^*$. For the base case $w = \epsilon$, we have $\overrightarrow{\mu}(\epsilon)F = F = F\mu(\epsilon)$. For the induction step, assume that $\overrightarrow{\mu}(w)F = F\mu(w)$ for some w. Then, for $\sigma \in \Sigma$, we have

$$\begin{aligned}
\overrightarrow{\mu}(\sigma w)F &= \overrightarrow{\mu}(\sigma)\overrightarrow{\mu}(w)F \\
&= \overrightarrow{\mu}(\sigma)F\mu(w) && \text{(IH)} \\
&= F\mu(\sigma)\mu(w) && \text{(definition of } \overrightarrow{\mu}\text{)} \\
&= F\mu(\sigma w).
\end{aligned}$$

By induction, we have $\overrightarrow{\mu}(w)F = F\mu(w)$ for all $w \in \Sigma^*$. □

The result for $\overleftarrow{\mu}$ is analogous, with the proof using exactly the same steps.

Lemma 4.6. *Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA, and let B be a matrix whose columns form a basis of $\mathcal{B}_\mathcal{A}$. If $\overleftarrow{\mathcal{A}} = (\overleftarrow{n}, \Sigma, \overleftarrow{\alpha}, \overleftarrow{\mu}, \overleftarrow{\beta})$ is constructed using B, then $B\overleftarrow{\mu}(w) = \mu(w)B$ for all $w \in \Sigma^*$.*

Proof. We will use induction on the length of $w \in \Sigma^*$. For the base case $w = \epsilon$, we have $B\overrightarrow{\mu}(\epsilon) = B = \mu(\epsilon)B$. For the induction step, assume that $B\overrightarrow{\mu}(w) = \mu(w)B$ for some w. Then, for $\sigma \in \Sigma$, we have

$$\begin{aligned}
B\overleftarrow{\mu}(w\sigma) &= B\overleftarrow{\mu}(w)\overleftarrow{\mu}(\sigma) \\
&= \mu(w)B\overleftarrow{\mu}(\sigma) && \text{(IH)} \\
&= \mu(w)\mu(\sigma)B && \text{(definition of } \overrightarrow{\mu}\text{)} \\
&= \mu(w\sigma)B.
\end{aligned}$$

By induction, we have $B\overrightarrow{\mu}(w) = \mu(w)B$ for all $w \in \Sigma^*$. □

This symmetry is not completely unexpected. Because we are working in a field, multiplication is commutative. If we look back at how we first defined the weight of a specific path – multiplying the weights in order – it now does not matter if we walk this path backwards, since the resulting weight will still be the same due to the commutativity.

Proposition 4.7. *Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA, and let F be a matrix whose rows form a basis of $\mathcal{F}_\mathcal{A}$. If $\overrightarrow{\mathcal{A}} = (\overrightarrow{n}, \Sigma, \overrightarrow{\alpha}, \overrightarrow{\mu}, \overrightarrow{\beta})$ is constructed using F, then $\overrightarrow{\mathcal{A}}$ is equivalent to $\mathcal{A}$.*

Proof. Recall that equivalence means that $[\![\overrightarrow{\mathcal{A}}]\!] = [\![\mathcal{A}]\!]$. For all $w \in \Sigma^*$, we have

$$\begin{aligned}
[\![\overrightarrow{\mathcal{A}}]\!](w) &= \overrightarrow{\alpha}\overrightarrow{\mu}(w)\overrightarrow{\beta} \\
&= \overrightarrow{\alpha}\overrightarrow{\mu}(w)F\beta && \text{(definition of } \overrightarrow{\beta}\text{)} \\
&= \overrightarrow{\alpha}F\mu(w)\beta && \text{(Lemma 4.5)} \\
&= \alpha\mu(w)\beta && \text{(definition of } \overrightarrow{\alpha}\text{)} \\
&= [\![\mathcal{A}]\!](w),
\end{aligned}$$

so $\overrightarrow{\mathcal{A}}$ is equivalent to $\mathcal{A}$. □

For $\overleftarrow{\mathcal{A}}$, the result is again analogous, but the proof is mirrored.

Proposition 4.8. *Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA, and let B be a matrix whose columns form a basis of $\mathcal{B}_\mathcal{A}$. If $\overleftarrow{\mathcal{A}} = (\overleftarrow{n}, \Sigma, \overleftarrow{\alpha}, \overleftarrow{\mu}, \overleftarrow{\beta})$ is constructed using B, then $\overleftarrow{\mathcal{A}}$ is equivalent to $\mathcal{A}$.*

Proof. For all $w \in \Sigma^*$, we have

$$\begin{aligned}
[\![\overleftarrow{\mathcal{A}}]\!](w) &= \overleftarrow{\alpha}\overleftarrow{\mu}(w)\overleftarrow{\beta} \\
&= \alpha B\overleftarrow{\mu}(w)\overleftarrow{\beta} && \text{(definition of } \overleftarrow{\alpha}\text{)} \\
&= \alpha\mu(w)B\overleftarrow{\beta} && \text{(Lemma 4.6)} \\
&= \alpha\mu(w)\beta && \text{(definition of } \overleftarrow{\beta}\text{)} \\
&= [\![\mathcal{A}]\!](w),
\end{aligned}$$

so $\overleftarrow{\mathcal{A}}$ is equivalent to $\mathcal{A}$. □

It now follows directly that the forward conjugate of the backward conjugate and the backward conjugate of the forward conjugate of a WFA are equivalent to the original WFA.

Corollary 4.9. *Let $\mathcal{A}$ be a WFA, and let $\widehat{\mathcal{A}}$ be the forward conjugate of $\overleftarrow{\mathcal{A}}$, or the backward conjugate of $\overrightarrow{\mathcal{A}}$. Then, $\widehat{\mathcal{A}}$ is equivalent to $\mathcal{A}$.*

Proof. If $\widehat{\mathcal{A}}$ is the forward conjugate of $\overleftarrow{\mathcal{A}}$, then first apply Lemma 4.6 and then Lemma 4.5. Otherwise, if $\widehat{\mathcal{A}}$ is the backward conjugate of $\overrightarrow{\mathcal{A}}$, first apply Lemma 4.5, followed by Lemma 4.6. □

For this section, it remains to show that $\widehat{\mathcal{A}}$ is not just equivalent to $\mathcal{A}$, but also minimal. We saw that the forward conjugate $\overrightarrow{\mathcal{A}}$ "trivialises" the initial vector α. This means that the backward space $\mathcal{B}_{\overrightarrow{\mathcal{A}}}$ of $\overrightarrow{\mathcal{A}}$ now contains all information of $\mathcal{A}$. It then makes sense to expect that the dimension of $\mathcal{B}_{\overrightarrow{\mathcal{A}}}$ is the minimal amount of states of a WFA that is equivalent to $\mathcal{A}$. The idea for the forward space of a backward conjugate is analogous. In the next theorem, we will formalise this idea. However, we need an intermediary result.

Lemma 4.10. *Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA, and let $\overleftarrow{\mathcal{A}} = (\overleftarrow{n}, \Sigma, \overleftarrow{\alpha}, \overleftarrow{\mu}, \overleftarrow{\beta})$ be the backward conjugate of $\mathcal{A}$. Let $\overleftarrow{\mathcal{A}}_b$ be the matrix whose rows form the set $\left\{\overleftarrow{\mu}(w)\overleftarrow{\beta} \mid w \in \Sigma^*\right\}$. Then, the rows of $\overleftarrow{\mathcal{A}}_b$ are linearly independent.*

Proof. Let $\overleftarrow{\mathcal{A}} = (\overleftarrow{n}, \Sigma, \overleftarrow{\alpha}, \overleftarrow{\mu}, \overleftarrow{\beta})$ be the backward conjugate of $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$, and let B be the matrix used to construct $\overleftarrow{\mathcal{A}}$. Then we have

$$\begin{aligned}
\operatorname{rank}(\overleftarrow{\mathcal{A}}_b) &= \dim\left\langle \overleftarrow{\mu}(w)\overleftarrow{\beta} \mid w \in \Sigma^* \right\rangle \\
&= \dim\left\langle B\overleftarrow{\mu}(w)\overleftarrow{\beta} \mid w \in \Sigma^* \right\rangle && (B \text{ has full column rank}) \\
&= \dim\left\langle \mu(w)B\overleftarrow{\beta} \mid w \in \Sigma^* \right\rangle && (\text{Lemma } 4.6) \\
&= \dim\langle \mu(w)\beta \mid w \in \Sigma^* \rangle && (\text{definition of } \overleftarrow{\beta}) \\
&= \overleftarrow{n}.
\end{aligned}$$

By construction, $\overleftarrow{\mathcal{A}}_b$ has $\overleftarrow{n}$ rows, and thus the rows of $\overleftarrow{\mathcal{A}}_b$ are linearly independent. □

Lemma 4.11. *Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA, and let $\overrightarrow{\mathcal{A}} = (\overrightarrow{n}, \Sigma, \overrightarrow{\alpha}, \overrightarrow{\mu}, \overrightarrow{\beta})$. Let $\mathcal{A}_f$ be the matrix whose rows form the set $\left\{\overrightarrow{\alpha}\overrightarrow{\mu}(w) \mid w \in \Sigma^*\right\}$. Then, the columns of $\overrightarrow{\mathcal{A}}_f$ are linearly independent.*

Proof. Let $\overrightarrow{\mathcal{A}} = (\overrightarrow{n}, \Sigma, \overrightarrow{\alpha}, \overrightarrow{\mu}, \overrightarrow{\beta})$ be the forward conjugate of $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$, and let F be the matrix used to construct $\overrightarrow{\mathcal{A}}$. Then we have

$$\begin{aligned}
\operatorname{rank}(\overrightarrow{\mathcal{A}}_f) &= \dim\left\langle \overrightarrow{\alpha}\overrightarrow{\mu}(w) \mid w \in \Sigma^* \right\rangle \\
&= \dim\left\langle \overrightarrow{\alpha}\overrightarrow{\mu}(w)F \mid w \in \Sigma^* \right\rangle && (F \text{ has full row rank}) \\
&= \dim\left\langle \overrightarrow{\alpha}F\mu(w) \mid w \in \Sigma^* \right\rangle && (\text{Lemma } 4.5) \\
&= \dim\langle \alpha\mu(w) \mid w \in \Sigma^* \rangle && (\text{definition of } \overrightarrow{\alpha}) \\
&= \overrightarrow{n}.
\end{aligned}$$

By construction, $\overrightarrow{\mathcal{A}}_f$ has $\overrightarrow{n}$ columns, and thus the columns of $\overrightarrow{\mathcal{A}}_f$ are linearly independent. □

Theorem 4.12. *Let $\mathcal{A}$ be a WFA. Then, the forward conjugate $\widehat{\mathcal{A}}$ of the backward conjugate $\overleftarrow{\mathcal{A}}$ and the backward conjugate $\widehat{\mathcal{A}}$ of the forward conjugate $\overrightarrow{\mathcal{A}}$ are minimal.*

Proof. **Intuitive idea.** Let $\overrightarrow{\mathcal{A}}_f \in R^{\Sigma^* \times \overrightarrow{n}}$ be the matrix such that the i-th row is given by $\overrightarrow{\alpha}\overrightarrow{\mu}(i)$. Similarly, let $\overrightarrow{\mathcal{A}}_b \in R^{\overrightarrow{n} \times \Sigma^*}$ be the matrix such that the j-th column is given by $\overrightarrow{\mu}(j)\overrightarrow{\beta}$. We have $\overrightarrow{\mathcal{A}}_f \cdot \overrightarrow{\mathcal{A}}_b = H_{\mathcal{A}}$. Since $\overrightarrow{\mathcal{A}}$ is a forward conjugate, $\overrightarrow{\mathcal{A}}_f$ has full column-rank. This means that the rank of $H_{\mathcal{A}}$ is determined by the rank of $\overrightarrow{\mathcal{A}}_b$. If we construct the backward conjugate of $\overrightarrow{\mathcal{A}}$, then the amount of states of this backward conjugate is equal to the rank of $\overrightarrow{\mathcal{A}}_b$. Thus, the amount of states of the backward conjugate of $\overrightarrow{\mathcal{A}}$ is equal to the rank of $H_{\mathcal{A}}$, which means that it is minimal by Theorem 3.11.

Formalisation. We will first prove the statement about the forward conjugate $\widehat{\mathcal{A}} = (\widehat{n}, \Sigma, \widehat{\alpha}, \widehat{\mu}, \widehat{\beta})$ of the backward conjugate $\overleftarrow{\mathcal{A}} = (\overleftarrow{n}, \Sigma, \overleftarrow{\alpha}, \overleftarrow{\mu}, \overleftarrow{\beta})$. Let F be a matrix used to construct $\widehat{\mathcal{A}}$. By the first part of Theorem 3.11, we have to show that $\widehat{n} = \operatorname{rank}(H_{\mathcal{A}})$. Let $\overleftarrow{\mathcal{A}}_f \in R^{\Sigma^* \times \overleftarrow{n}}$ be the matrix such that the i-th row is given by $\overleftarrow{\alpha}\overleftarrow{\mu}(i)$, and let $\overleftarrow{\mathcal{A}}_b \in R^{\overleftarrow{n} \times \Sigma^*}$ be the matrix such that the j-th column is given by $\overleftarrow{\mu}(j)\overleftarrow{\beta}$. By Lemma 4.10, the rows of $\overleftarrow{\mathcal{A}}_b$ are linearly independent. We have

$$
\begin{aligned}
\widehat{n} &= \operatorname{rank}(F) \\
&= \operatorname{rank}(F\overleftarrow{\mathcal{A}}_b) && \text{(the rows of } \overleftarrow{\mathcal{A}}_b \text{ are linearly independent)} \\
&= \dim\left\langle \overleftarrow{\alpha}\overleftarrow{\mu}(w)\overleftarrow{\mathcal{A}}_b \mid w \in \Sigma^* \right\rangle && \text{(definition of } F\text{)} \\
&= \operatorname{rank}(\overleftarrow{\mathcal{A}}_f \overleftarrow{\mathcal{A}}_b) && \text{(definition of } \overleftarrow{\mathcal{A}}_f\text{)} \\
&= \operatorname{rank}(H_{\overleftarrow{\mathcal{A}}}) && (\overleftarrow{\mathcal{A}}_f \overleftarrow{\mathcal{A}}_b = H_{\overleftarrow{\mathcal{A}}}) \\
&= \operatorname{rank}(H_{\mathcal{A}}) && (H_{\overleftarrow{\mathcal{A}}} = H_{\mathcal{A}}).
\end{aligned}
$$

The proof for the statement about the backward conjugate of $\widehat{\mathcal{A}} = (\widehat{n}, \Sigma, \widehat{\alpha}, \widehat{\mu}, \widehat{\beta})$ of the forward conjugate $\overrightarrow{\mathcal{A}} = (\overrightarrow{n}, \Sigma, \overrightarrow{\alpha}, \overrightarrow{\mu}, \overrightarrow{\beta})$ is, as expected, analogous. Let $\overrightarrow{\mathcal{A}}_f \in R^{\Sigma^* \times \overrightarrow{n}}$ be the matrix such that the i-th row is given by $\overrightarrow{\alpha}\overrightarrow{\mu}(i)$, and let $\overrightarrow{\mathcal{A}}_b \in R^{\overrightarrow{n} \times \Sigma^*}$ be the matrix such that the j-th column is given by $\overrightarrow{\mu}(j)\overrightarrow{\beta}$. By Lemma 4.11, the columns of $\overrightarrow{\mathcal{A}}_b$ are linearly independent.

We have

$$
\begin{aligned}
\widehat{n} &= \operatorname{rank}(F) \\
&= \operatorname{rank}(\overrightarrow{\mathcal{A}}_f B) && \text{(the columns of } \overrightarrow{\mathcal{A}}_b \text{ are linearly independent)} \\
&= \dim \left\langle \overrightarrow{\mathcal{A}}_f \overrightarrow{\mu}(w) \overrightarrow{\beta} \mid w \in \Sigma^* \right\rangle && \text{(definition of } B\text{)} \\
&= \operatorname{rank}(\overrightarrow{\mathcal{A}}_f \overrightarrow{\mathcal{A}}_b) && \text{(definition of } \overrightarrow{\mathcal{A}}_f\text{)} \\
&= \operatorname{rank}(H_{\overrightarrow{\mathcal{A}}}) && (\overrightarrow{\mathcal{A}}_f \overrightarrow{\mathcal{A}}_b = H_{\overrightarrow{\mathcal{A}}}) \\
&= \operatorname{rank}(H_{\mathcal{A}}) && (H_{\overrightarrow{\mathcal{A}}} = H_{\mathcal{A}}).
\end{aligned}
$$

This completes the proof. □

4.2 Minimisation over the integers

In the previous section, we described how to minimise WFAs over fields. However, $\mathbb{Z}$ is not a field, so it is not possible to directly use this method to minimise a WFA over $\mathbb{Z}$. Despite this, some modifications to the standard minimisation algorithms do allow us to minimise WFAs with integer-valued semantics, using only integer weights, even if the original WFA uses weights in $\mathbb{Q}$. Such an algorithm was defined in [5], and we study it here. This algorithm can be seen as an implementation of the fact that $\mathbb{Q}$ is a Fatou extension of $\mathbb{Z}$ [3]. Formally, this means that any formal power series of $\mathbb{Z}$ which is $\mathbb{Q}$-rational is also $\mathbb{Z}$-rational. In WFA terms, this means for every WFA $\mathcal{A}_1$ over $\mathbb{Q}$ such that $[\![\mathcal{A}]\!](w) \in \mathbb{Z}$ for every w, there exists a WFA $\mathcal{A}_2$ over $\mathbb{Z}$ that is equivalent to $\mathcal{A}_1$.

Because we are dealing with $\mathbb{Z}$, which is a principal ideal domain, but not a field, we have to make a distinction between the forward and backward space, and the forward and backward *module*. The definition of the forward and backward module is analogous to the forward and backward space, except that we use a module instead of a vector space. In other words, the forward and backward module is the ring version of the forward and backward space. We use $\langle \alpha\mu(w) \mid w \in \Sigma^* \rangle_{\mathbb{Q}}$ and $\langle \mu(w)\beta \mid w \in \Sigma^* \rangle_{\mathbb{Q}}$ to respectively denote the forward and backward space over $\mathbb{Q}$, and $\langle \alpha\mu(w) \mid w \in \Sigma^* \rangle_{\mathbb{Z}}$ and $\langle \mu(w)\beta \mid w \in \Sigma^* \rangle_{\mathbb{Z}}$ to respectively denote the forward and backward module over $\mathbb{Z}$.

The idea of the algorithm is as follows.

1. Construct the backward conjugate $\overleftarrow{\mathcal{A}}$ of $\mathcal{A}$. It now holds that for all $w \in \Sigma^*$, for every element s of the vector $\overleftarrow{\alpha}\,\overleftarrow{\mu}(w)$, there exists a $v \in \Sigma^*$ such that $s = [\![\mathcal{A}]\!](v)$.
2. Find a basis of the forward space of $\overleftarrow{\mathcal{A}}$. If we find some $w \in \Sigma^*$ such that $\overleftarrow{\alpha}\,\overleftarrow{\mu}(w) \notin \mathbb{Z}^n$, then $[\![\mathcal{A}]\!]$ is not strictly integer valued, and we have found a counterexample.
3. Find a set of words W_F such that $\{\alpha\mu(w) \mid w \in W_F\}$ spans the forward module of $\overleftarrow{\mathcal{A}}$, using the basis from the previous step as a starting point. Again, check if every vector is strictly integer valued to find a counterexample.
4. Use the set W_F of the previous point to construct an integer basis of the forward module of $\overleftarrow{\mathcal{A}}$.
5. Use the basis of the forward module to construct the forward conjugate of $\overleftarrow{\mathcal{A}}$. This forward conjugate is minimal, uses only integer weights, and is equivalent to $\mathcal{A}$.

A pseudo-code version of the algorithm is shown in Figure 4.2. Steps 2 and 3 have been separated into the function `compute_Z_generators`.

One might wonder why it is necessary to first find a basis of the forward space of $\overleftarrow{\mathcal{A}}$, and only then find a basis of the forward module. This order ensures that each intermediate module has the same rank, which makes it possible to put a polynomial upper bound on

the amount of iterations it takes to find a forward generating set of the forward module. It is not known whether there exist other search orders which ensure a polynomial upper bound [5].

First of all, we want to formalise the argument made in step 1, which is that all elements of the vectors $\alpha\mu(w)$ while finding the basis of the forward space of $\overleftarrow{\mathcal{A}}$ have the form $[\![\mathcal{A}]\!](v)$.

Proposition 4.13. *Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA, and let $\overleftarrow{\mathcal{A}} = (\overleftarrow{n}, \Sigma, \overleftarrow{\alpha}, \overleftarrow{\mu}, \overleftarrow{\beta})$ be its backward conjugate. Then, for every $w \in \Sigma^*$, we have $\left(\overleftarrow{\alpha}\overleftarrow{\mu}(w)\right)_i \in [\![\mathcal{A}]\!](\Sigma^*)$ for every $i \leq \overleftarrow{n}$.*

Proof. Let the WFA $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be given, and assume that its backward conjugate $\overleftarrow{\mathcal{A}} = (\overleftarrow{n}, \Sigma, \overleftarrow{\alpha}, \overleftarrow{\mu}, \overleftarrow{\beta})$ was constructed using the matrix B. Fix $w \in \Sigma^*$. We have $\overleftarrow{\alpha}\overleftarrow{\mu}(w) = \alpha B\overleftarrow{\mu}(w) = \alpha\mu(w)B$, where we used Lemma 4.6 for the last equality. Recall that the columns of B have the form $\mu(v)\beta$ for $v \in \Sigma^*$. This means that the i-th entry of $\alpha\mu(w)B$ is given by $\alpha\mu(w)\mu(v)\beta = \alpha\mu(wv)\beta = [\![\mathcal{A}]\!](wv) \in [\![\mathcal{A}]\!](\Sigma^*)$. □

We can use this to not only find that the semantics of $\mathcal{A}$ are not strictly integer valued, but also to find an explicit word w such that $[\![\mathcal{A}]\!](w) \notin \mathbb{Z}$. If we remember which words were used to construct the basis of the backward space of $\mathcal{A}$, then the i-th entry of $\overleftarrow{\alpha}\overleftarrow{\mu}(w)$ is given by $[\![\mathcal{A}]\!](wv)$, where v is the word such that the i-th column of B is $\mu(v)\beta$. This is the reason why W_B, the words used to construct B, is passed as an argument to `compute_ℤ_generators`.

Remark 4.14. Lines 4 to 6 in `compute_ℤ_generators` are not included in the original algorithm in [5]. However, they are necessary. If $\mathcal{A}$ is an automaton such that $[\![\mathcal{A}]\!]$ is integer valued except for the empty word, then this only be detected in lines 4 to 6, and not in the rest of the algorithm.

It is not necessary to directly prove that not finding a counterexample during the algorithm implies that no such counterexample exists. The construction of the final WFA uses only integer weights, which means that its semantics must also be strictly integer valued. Because this WFA is equivalent to the original WFA, the semantics of the original WFA must be strictly integer valued as well.

Before we state and prove the final theorem, we will prove that `compute_ℤ_generators` terminates in polynomial time, with the respect to the length of the encoding of $\overleftarrow{\mathcal{A}}$. We have already shown that lines 8 to 13 terminate in polynomial time in Lemma 4.2. Thus, it remains to show that lines 15 to 20 terminate in polynomial time. We cannot use the same argument as we used in Lemma 4.2 here, because proper sub-modules can have the same rank. For example, $8\mathbb{Z} \subsetneq 4\mathbb{Z} \subsetneq 2\mathbb{Z} \subsetneq \mathbb{Z}$ are all modules of rank 1. However, we can put an upper bound of the length of a chain of sub-modules of the same rank, which is exactly why we first find a basis for the forward space of $\overleftarrow{\mathcal{A}}$.

Proposition 4.15. *Let $n, k \in N$. Let R be a PID, and $M = \langle v_1, v_2, \ldots, v_m \rangle$ be a sub-module of R^n. Let A be the $n \times m$ matrix whose i-th column is v_i. Let $M \subsetneq M_1 \subsetneq M_2 \subsetneq \cdots \subsetneq M_k$ be a chain of sub-modules of R^n, all having the same rank r. Then k is bounded by the number of prime factors of the greatest common divisor of the $r \times r$ minors of A.*

For a detailed proof, see appendix A of [5].

Since $\mathbb{Z}$ is a PID, we can use this proposition to bound the length of a chain of $\mathbb{Z}$-modules, using the fact that the number of prime factors of an integer n is bound by $\log_2(n)$.

It is now time for the final result of this thesis.

Theorem 4.16. *The algorithm `compute_Z_basis` in Figure 4.2 returns, given a WFA $\mathcal{A}$ over $\mathbb{Q}$, a minimal equivalent WFA over $\mathbb{Z}$ if $[\![\mathcal{A}]\!]$ is integer valued, and otherwise a word w such that $[\![\mathcal{A}]\!](w) \notin \mathbb{Z}$, in polynomial time with respect to the length of the binary encoding of $\mathcal{A}$.*

Proof. **Minimising over $\mathbb{Z}$.** Let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA over $\mathbb{Q}$, and assume that $[\![\mathcal{A}]\!]$ is integer valued. By Lemma 4.2, lines 24-31 compute a backward conjugate $\overleftarrow{\mathcal{A}}$ of $\mathcal{A}$. The algorithm then proceeds to `compute_Z_generators`. By Lemma 4.13 and the fact that $[\![\mathcal{A}]\!]$ is integer valued, it follows that the condition on lines 5, 10, and 17 will never evaluate to `True`. When the algorithm reaches line 21, by Lemma 4.2, F spans the forward module of $\overleftarrow{\mathcal{A}}$. Using Lemma 2.4, we can find a matrix $\widehat{F}$ whose rows form a $\mathbb{Z}$-basis of the forward module of $\overleftarrow{\mathcal{A}}$, and whose rows also form a $\mathbb{Q}$-basis of the forward space of $\overleftarrow{\mathcal{A}}$. Thus, computing the forward conjugate $\widehat{\mathcal{A}}$ of $\overleftarrow{\mathcal{A}}$ using $\widehat{F}$ gives a minimal WFA that is equivalent to $\mathcal{A}$ by Lemma 4.12. Furthermore, since the rows of $\widehat{F}$ form an integer basis, the equations in Definition 4.3 have integer solutions, so $\widehat{\mathcal{A}}$ is a WFA over $\mathbb{Z}$.

Finding a counterexample. Now, let $\mathcal{A} = (n, \Sigma, \alpha, \mu, \beta)$ be a WFA over $\mathbb{Q}$, and let there be some $w \in \Sigma^*$ such that $[\![\mathcal{A}]\!](w) \notin \mathbb{Z}$. Since the algorithm returns a WFA over $\mathbb{Z}$ on line 39, which is not possible when $[\![\mathcal{A}]\!]$ is not integer valued, the algorithm must return on line 36, and thus have found some w, σ, i such that $\overleftarrow{\alpha}\,\overleftarrow{\mu}(w\sigma)_i \notin \mathbb{Z}$. Since $\overleftarrow{\alpha}\,\overleftarrow{\mu}(w\sigma) = \alpha\mu(w\sigma)B$, where B is the matrix used to compute the backward conjugate $\overleftarrow{\mathcal{A}}$, and the i-th row of B is equal to $\mu(W_B[i])\beta$, it follows that $\overleftarrow{\alpha}\,\overleftarrow{\mu}(w\sigma)_i = \alpha\mu(w\sigma)\mu(W_B[i])\beta = \alpha\mu(w\sigma W_B[i])\beta = [\![\mathcal{A}]\!](w\sigma W_B[i]) \notin \mathbb{Z}$, and thus $w\sigma W_B[i]$ is a counterexample.

Time complexity. By Lemma 4.2, we only have to prove that the lines 15-20 terminate in polynomial time. In particular, this means that the amount of iterations of the `while` loop on line 15 must be polynomial. By Lemma 4.2, we have that F is an $m \times n$ matrix with rank m when we first reach line 15. Thus, the $\langle F_1 \rangle_{\mathbb{Z}}, \langle F_2 \rangle_{\mathbb{Z}}, \ldots, \langle F_k \rangle_{\mathbb{Z}}$ form a chain of $\mathbb{Z}$-modules of rank m, where each subsequent F_i is the matrix F at the beginning of each iteration of the `while` loop. By Lemma 2.4, and in particular equation 2.1, it follows that $d_1 \cdot d_2 \cdot \cdots \cdot d_m$ is the greatest common divisor of all $m \times m$ minors of F_1. Let

K be the least upper bound of the absolute values of the entries of F. Using Lemma 2.5, it follows that $d_1 \cdot d_2 \cdot \cdots \cdot d_m \leq K^m m^{\frac{m}{2}}$, where $d_1, d_2, \ldots, d_m$ are the diagonal entries of the Smith Normal Form of F. Then, according to Lemma 4.15, the length of the chain $\langle F_1 \rangle_{\mathbb{Z}}, \langle F_2 \rangle_{\mathbb{Z}}, \ldots, \langle F_k \rangle_{\mathbb{Z}}$ is bound the number of prime factors of $K^m m^{\frac{m}{2}}$, which is bound by $\log_2(K^m m^{\frac{m}{2}})$. Since the length of words W_F at the start of the while loop is at most n, and the rows of F have the form $\alpha\mu(w)$ with $w \in W_F$, it follows that theabsolute value of the entries of F are bounded by $n^{n-1}L^n$, where L is the least upper bound of the entries of α, β, and $\mu(\sigma)$ for $\sigma \in \Sigma$. Thus, the amount of iterations of the `while` loop is bounded by $\log_2(K^m m^{\frac{m}{2}}) \leq \log_2((n^{n-1}L^n)^n n^{\frac{n}{2}}) = n(n-\frac{1}{2})\log_2(n) + n^2\log_2(L)$, since $m \leq n$. Since n and $\log_2(L)$ are both polynomial with respect to the length of the binary encoding of $\mathcal{A}$, it follows that the `while` loop iterates at most a polynomial amount of times.

Finally, we must ensure that we can find solutions to matrix equations in Definition 4.3 and 4.4 in polynomial time. While the entries of F and B are exponential with respect to L, finding a solution to these equations is polynomial with respect to the size of the encoding of L [20], which is logarithmic with respect to L. □

4.2.1 Application of the algorithm

In this section, we will apply the algorithm on the WFA $\mathcal{A} = (3, \{a, b\}, \alpha, \mu, \beta)$, where

$$\alpha = \begin{pmatrix} \frac{1}{2} & \frac{3}{2} & 0 \end{pmatrix} \qquad \mu(a) = \begin{pmatrix} -1 & -4 & 0 \\ 1 & 2 & 0 \\ \frac{1}{3} & 1 & -2 \end{pmatrix} \qquad \mu(b) = \begin{pmatrix} 0 & 6 & 0 \\ 1 & -1 & 0 \\ 2 & 2 & -\frac{1}{2} \end{pmatrix} \qquad \beta = \begin{pmatrix} 0 \\ 2 \\ -\frac{1}{2} \end{pmatrix}.$$

This WFA does not represent anything particularly interesting, but it does cover a quite complete set of cases while performing the algorithm, so it works well as an example.

First, we need to find a basis of the backward space of $\mathcal{A}$. We start by setting

$$W_B = \{\epsilon\} \qquad B = \beta = \begin{pmatrix} 0 \\ 2 \\ -\frac{1}{2} \end{pmatrix},$$

and we need to check the loop condition, namely if there is some $w \in W_B$ and $\sigma \in \{a, b\}$ such that $\mu(\sigma w)\beta \notin \langle B \rangle_{\mathbb{Q}}$. In this case for $w = \epsilon$ and $\sigma = a$, we see that

$$\mu(a)\beta = \begin{pmatrix} -1 & -4 & 0 \\ 1 & 2 & 0 \\ \frac{1}{3} & 1 & -2 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 2 \\ -\frac{1}{2} \end{pmatrix} = \begin{pmatrix} -8 \\ 4 \\ 3 \end{pmatrix}$$

is linearly independent from B, so we update

$$W_B = \{\epsilon, a\} \qquad B = \begin{pmatrix} 0 & -8 \\ 2 & 4 \\ -\frac{1}{2} & 3 \end{pmatrix}.$$

Next, we find that for $w = \epsilon$ and $\sigma = b$, we have

$$\mu(b)\beta = \begin{pmatrix} 0 & 6 & 0 \\ 1 & -1 & 0 \\ 2 & 2 & -\frac{1}{2} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 2 \\ -\frac{1}{2} \end{pmatrix} = \begin{pmatrix} 12 \\ -2 \\ \frac{17}{4} \end{pmatrix},$$

is again linearly independent from B, so we now get

$$W_B = \{\epsilon, a, b\} \qquad B = \begin{pmatrix} 0 & -8 & 12 \\ 2 & 4 & -2 \\ -\frac{1}{2} & 3 & \frac{17}{4} \end{pmatrix}.$$

Since B has full rank, we know that the loop will terminate and we continue with the next part of the algorithm, which is computing the backward conjugate of $\mathcal{A}$. Recall that this requires finding matrices $\overleftarrow{\alpha}$, $\overleftarrow{\mu}(a)$, $\overleftarrow{\mu}(b)$ and $\overleftarrow{\beta}$ such that

$$\overleftarrow{\alpha} = \alpha B \qquad B\overleftarrow{\mu}(a) = \mu(a)B \qquad B\overleftarrow{\mu}(b) = \mu(b)B \qquad B\overleftarrow{\beta} = \beta.$$

With some linear algebra, we find

$$\overleftarrow{\alpha} = \begin{pmatrix} 3 & 2 & 3 \end{pmatrix} \qquad \overleftarrow{\mu}(a) = \begin{pmatrix} 0 & -\frac{2}{9} & \frac{13}{3} \\ 1 & -\frac{1}{3} & -\frac{1}{2} \\ 0 & -\frac{8}{9} & -\frac{2}{3} \end{pmatrix} \qquad \overleftarrow{\mu}(b) = \begin{pmatrix} 0 & \frac{4}{39} & \frac{11}{13} \\ 0 & \frac{-40}{13} & \frac{201}{52} \\ 1 & -\frac{2}{39} & \frac{41}{26} \end{pmatrix} \qquad \overleftarrow{\beta} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

We continue with finding a basis of the forward space of $\overleftarrow{\mathcal{A}}$. Using steps analogous to finding the backward space of $\mathcal{A}$, we find that

$$W_F = \{\epsilon, a\} \qquad F = \begin{pmatrix} 3 & 2 & 3 \\ 2 & -4 & 10 \end{pmatrix},$$

and go to finding a basis of the forward module of $\overleftarrow{\mathcal{A}}$. For $w = \epsilon$ and $\sigma = b$, we find that

$$\alpha\mu(\beta) = \begin{pmatrix} 3 & -6 & 15 \end{pmatrix}$$

cannot be written as an integer combination of the rows of F, so we update

$$W_F = \{\epsilon, a, b\} \qquad F = \begin{pmatrix} 3 & 2 & 3 \\ 2 & -4 & 10 \\ 3 & -6 & 15 \end{pmatrix}.$$

Now, for all $w \in W_F$ and $\sigma \in \Sigma$, $\alpha\mu(w\sigma)$ can be written as a linear combination of rows of F. For example, for $w = a$ and $\sigma = b$ we have

$$\alpha\mu(ab) = \begin{pmatrix} 10 & 12 & 2 \end{pmatrix} = 4 \cdot \begin{pmatrix} 3 & 2 & 3 \end{pmatrix} - \begin{pmatrix} 2 & -4 & 10 \end{pmatrix}.$$

Now, we can use the Hermite Normal Form of F to find an integer basis of the forward module of $\overleftarrow{\mathcal{A}}$. We have

$$\mathrm{HNF}(F) = \begin{pmatrix} 1 & -6 & -7 \\ 0 & 8 & -12 \\ 0 & 0 & 0 \end{pmatrix},$$

and the final automaton is thus given by conjugating $\overleftarrow{\mathcal{A}}$ with the matrix consisting of the non-zero rows of $\mathrm{HNF}(F)$, namely the matrix

$$\begin{pmatrix} 1 & -6 & -7 \\ 0 & 8 & -12 \end{pmatrix}.$$

With some basis linear algebra, we find that the final automaton is given by

$$\overleftarrow{\alpha} = \begin{pmatrix} 3 & -2 \end{pmatrix} \qquad \overleftarrow{\mu}(a) = \begin{pmatrix} 6 & -4 \\ 8 & -5 \end{pmatrix} \qquad \overleftarrow{\mu}(b) = \begin{pmatrix} -7 & 3 \\ -12 & 6 \end{pmatrix} \qquad \overleftarrow{\beta} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

4.2.2 Implementation in Python

An implementation of the algorithm of the previous section has been made in Python, making use of the SymPy library [15]. The code of this library is available on GitHub[1]. It defines a class `WeightedAutomaton`, which uses the matrix representation of a WFA. This class provides several methods for working with WFAs, such as computing the weights of words and minimisation over the integers. Currently, the implementation supports WFAs over $\mathbb{Z}$ and $\mathbb{Q}$, but it can be extended to other semirings. However, note that minimisation only works over fields, and minimisation over the integers only works for WFAs over $\mathbb{Z}$ or $\mathbb{Q}$.

The repository contains some examples on how to use the minimisation over the integers. The output of this procedure is given as a WFA using SymPy matrices, which are symbolic and interpreted. While the implementation does attain a polynomial time complexity, the actual running time is limited by the speed of the SymPy library, which is not particularly fast for large matrices, especially in the initialisation of matrices. Currently, it is not practical to use the implementation for WFAs with over 100 states. However, after minimisation it is possible to use the much faster numerical NumPy matrices for weight calculations. This makes the program applicable in practical situations where WFAs without too many states are used and performance is important. Lowering the amount of states of a WFA can significantly reduce the complexity of weight calculations. It is not useful to also implement the minimisation itself for numerical matrices, as floating point rounding errors arise quickly.

[1] https://github.com/JLaumen/weighted-automata

```
def compute_ℤ_generators(W_B, n, Σ, α, μ, β):
    W_F = {ε}
    F = {α}
    for i in range(n):
        if α_{i+1} ∉ ℤ:
            return W_B[i]
    # Finding a basis of the forward space
    while there is (w, σ) ∈ W_F × Σ such that αμ(wσ) ∉ ⟨αμ(v) | v ∈ W_F⟩_ℚ:
        for i in range(n):
            if α_{i+1} ∉ ℤ: # Check if integer valued
                return concat(wσ, W_B[i])
        W_F = W_F ∪ {wσ}
        F = F ∪ {αμ(wσ)}
    # Finding a basis of the forward module
    while there is (w, σ) ∈ W_F × Σ such that αμ(wσ) ∉ ⟨αμ(v) | v ∈ W_F⟩_ℤ:
        for i in range(n):
            if α_{i+1} ∉ ℤ: # Check if integer valued
                return concat(wσ, W_B[i])
        W_F = W_F ∪ {wσ}
        F = F ∪ {αμ(wσ)}
    S, Â, T = smith_normal_form(F)
    return SÂ

def compute_ℤ_automaton(n, Σ, α, μ, β):
    W_B = {ε}
    B = {β}
    # Finding a basis of the backward space
    while there is (w, σ) ∈ W_B × Σ such that μ(σw)β ∉ ⟨μ(v)β | v ∈ W_B⟩_ℚ:
        W_B = W_B ∪ {σw}
        B = B ∪ {μ(σw)β}
    # Conjugate the automaton with B
    Ā = (n̄, Σ, ᾱ, μ̄, β̄)
    F = compute_ℤ_generators(W_B, n̄, Σ, ᾱ, μ̄, β̄)
    if F ∈ Σ*:
        # Return counter-example
        return F
    S, Â, T = smith_normal_form(F)
    # Conjugate Ā with F
    return (n̂, Σ, α̂, μ̂, β̂)
```

Figure 4.2: Algorithm to minimise a WFA over the integers [5].

Chapter 5

Related Work

Minimisation of WFAs over fields goes back to [22], but no explicit algorithm was provided. Explicit procedures were first introduced in [4] and [9], and later improved in [13]. The algorithms and proofs in this paper for minimisation over fields closely resemble [12], although this thesis provides a more intuitive description in addition to the proofs, and also provides an extensive introduction to WFAs, which is not included in most modern papers. A proof of Fliess' theorem is also given in [2], but no concrete example of applying Fliess' theorem is given.

Minimisation over the integers has been defined in [5], which uses this minimisation in order to learn integer WFAs. Learning WFAs over PIDs, including the integers, has been studied in the past [25], but [5] is the first paper to do so in polynomial time. In this paper, we focus on the minimisation itself, rather than its application in learning WFAs. We also work out a concrete example of the integer minimisation algorithm.

Weighted finite automata have been practically implemented in the past[1], but as far as we know, the library provided in this thesis is the first to implement minimisation over either fields or the integers, and is also the first to use symbolic operations instead of numerical methods, eliminating the problem of floating point rounding errors when using rational weights.

[1] https://github.com/jasperhoogland/jautomata

Chapter 6

Conclusion

In this thesis, we have discussed the theory of weighted finite automata, with an emphasis on the minimisation of these WFAs over the integers. In Chapter 3, we started by approaching WFAs intuitively, and used this to arrive at a formal definition. We then defined the product construction for WFAs and proved its correctness. We finished Chapter 3 by studying one of the main results of the theory of WFAs, and used this to motivate the minimisation of WFAs over the integers.

In Chapter 4, we defined and proved a classical minimisation algorithm for WFAs over fields. We then adapted this algorithm to work with integers, using the algorithm defined in [5], and proved the correctness of this adaptation. Finally, we discussed an implementation of this algorithm in Python, which is the main contribution of this thesis.

Since WFAs are often used in computationally expensive applications such as modelling neural networks [26], reducing the computational complexity of weight calculations can be of great importance. The implementation provided with this thesis can be used in several different practical applications, while the thesis itself serves as an overview and motivation of the theory behind it.

6.1 Future work

The algorithm for minimising WFAs over integers can be generalised to minimising WFAs over PIDs and their corresponding quotient fields. Both Lemma 2.4 and Proposition 4.15 do work for arbitrary PIDs, and the quotient fields of these WFAs can be used to work with the forward and backward spaces of these WFAs. Future work can include generalising our implementation for arbitrary PIDs. However, minisation over the integers already covers the most important use case.

What remains open is the time complexity of minimisation over arbitrary PIDs. While Lemma 4.15 does provide an upper bound with respect to the amount of prime factors, it is not clear how this relates to the length of the encoding of a WFA. More work is needed to study the relationship between the amount of prime factors and the length of the encoding.

Finally, [5] states that it is unknown whether there are other polynomial time search orders for finding a basis of the forward module of a WFA. While this question is not directly as interesting as the future work stated in the previous paragraphs due to a polynomial search order already existing, other search orders might provide more insight in the time complexity of minimisation of WFAs over arbitrary PIDs.

Bibliography

[1] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Formal analysis of online algorithms. In *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 2011.

[2] Borja Balle and Mehryar Mohri. Learning weighted automata. In *CAI*, volume 9270 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2015.

[3] Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*, volume 129 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2010.

[4] Jean Berstel and Christophe Reutenauer. *Rational series and their languages*, volume 12 of *EATCS monographs on theoretical computer science*. Springer, 1988.

[5] Alex Buna-Marginean, Vincent Cheval, Mahsa Shirmohammadi, and James Worrell. On learning polynomial recursive programs. *Proc. ACM Program. Lang.*, 8(POPL):1001–1027, 2024.

[6] Manfred Droste and Dietrich Kuske. Weighted automata. In *Handbook of Automata Theory (I.)*, pages 113–150. European Mathematical Society Publishing House, Zürich, Switzerland, 2021.

[7] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[8] Michel Fliess. Matrices de hankel. *J. Math. Pures Appl*, 53(9):197–222, 1974.

[9] David Gillman and Michael Sipser. Inference and minimization of hidden markov chains. In *COLT*, pages 147–158. ACM, 1994.

[10] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007.

[11] Karel Culík II and Jarkko Kari. Image compression using weighted finite automata. *Comput. Graph.*, 17(3):305–313, 1993.

[12] Stefan Kiefer. Notes on equivalence and minimization of weighted automata. *CoRR*, abs/2009.01217, 2020.

[13] Stefan Kiefer and Björn Wachter. Stability and complexity of minimising probabilistic automata. In *ICALP (2)*, volume 8573 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 2014.

[14] Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Int. J. Algebra Comput.*, 4(3):405–426, 1994.

[15] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[16] Mehryar Mohri, Cyril Allauzen, and Michael Riley. Statistical modeling for unit selection in speech synthesis. In *ACL*, pages 55–62. ACL, 2004.

[17] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted automata in text and speech processing. *CoRR*, abs/cs/0503077, 2005.

[18] Morris Newman. The smith normal form. *Linear Algebra and its Applications*, 254(1):367–381, 1997. Proceeding of the Fifth Conference of the International Linear Algebra Society.

[19] Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science. Springer, 1978.

[20] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[21] Gijs Schröder, Inge M. N. Wortel, and Johannes Textor. Implementing immune repertoire models using weighted finite state machines. *CoRR*, abs/2308.03637, 2023.

[22] Marcel Paul Schützenberger. On the definition of a family of automata. *Inf. Control.*, 4(2-3):245–270, 1961.

[23] Henry John Stephen Smith and James Joseph Sylvester. I. on systems of linear indeterminate equations and congruences. *Proceedings of the Royal Society of London*, 11:86–89, 1862.

[24] Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. Comput.*, 21(2):216–227, 1992.

[25] Gerco van Heerdt, Clemens Kupke, Jurriaan Rot, and Alexandra Silva. Learning weighted automata over principal ideal domains. In *FoSSaCS*, volume 12077 of *Lecture Notes in Computer Science*, pages 602–621. Springer, 2020.

[26] Zeming Wei, Xiyue Zhang, and Meng Sun. Extracting weighted finite automata from recurrent neural networks for natural languages. In *ICFEM*, volume 13478 of *Lecture Notes in Computer Science*, pages 370–385. Springer, 2022.