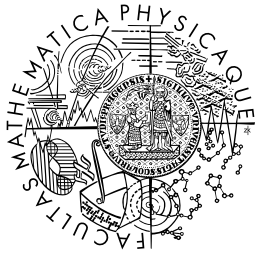


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Viktória Hurtišová

Representing Images by Weighted Finite Automata

Department of Algebra

Supervisor of the bachelor thesis: doc. Mgr. Štěpán Holub, Ph.D.

Study programme: Mathematics

Study branch: Mathematics for Information Technologies

Prague 2022

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Title: Representing Images by Weighted Finite Automata

Author: Viktória Hurtišová

Department: Department of Algebra

Supervisor: doc. Mgr. Štěpán Holub, Ph.D., Department of Algebra

Abstract: The goal of this thesis is to introduce weighted finite automata (WFA) as a means of representing multi-resolution raster images. We explain the basic concepts of weighted finite automata. Then we describe an encoding algorithm that converts an image into a WFA and a decoding algorithm that can generate back the original image. We then provide our implementation of the encoding and decoding algorithms.

Keywords: automaton WFA image

Contents

Introduction

The idea of representing images with weighted finite automata (WFA) was first introduced by Culik II, Karhumäki and Kari in the 1990s [1, 2], who have also shown that this representation has many applications in image manipulation and compression. We were mainly interested in representing multi-resolution images by WFA. The main advantage of this representation is that images in different resolutions can be generated from a single automaton. Simple inference algorithms exist for constructing WFA representing given images. These algorithms are given a finite-resolution raster image as an input and produce a relatively small WFA as an output. The original image can be then efficiently generated from the WFA.

The goal of this thesis is to introduce the basic concepts of weighted finite automata and then present some of the algorithms for encoding images into WFA and for decoding those automata back into images. Moreover, we present our implementation of the encoding and decoding algorithms.

We start in chapter 1 by introducing an addressing scheme of pixels using words over a four-letter alphabet together with the concept of multi-resolution images. In chapter 2 we introduce weighted finite automata with their properties. In this chapter we also explain how we can represent images with WFA. In chapter 3 we describe an algorithm for generating images from WFA and then we describe an algorithm for encoding a raster image into a WFA. In chapter 4 we will cover technical problems we encountered when implementing the encoding and decoding algorithms, and propose solutions for those problems.

1. Image Types and Their Representation

In this chapter, we give a formal definition of images with finite-resolution and multi-resolution images, as well as an addressing scheme of pixels using words over a four-letter alphabet. The addressing scheme will be used in all subsequent chapters.

Most of the definitions are taken from [3], [4] and [5]. Also, the overall structure of this chapter is inspired by these articles.

1.1 Finite Resolution Images

Definition 1. *A **semiring** R is a non-empty set on which are defined operations of addition and multiplication and two constants* 0 *and* 1 *such that:*

1. $(R, +, 0)$ *is a commutative monoid with an identity element* 0,
2. $(R, \cdot, 1)$ *is a monoid with identity element* $1 \neq 0$,
3. $a(b + c) = ab + ac$ *and* $(a + b)c = ac + bc$ *for all* $a, b, c \in R$,
4. $0a = 0 = a0$ *for all* $a \in R$.

The semiring R is commutative if the monoid $(R, \cdot, 1)$ is commutative.

Definition 2. *A **finite-resolution** image is a digitalized picture that consists of a rectangular array of $w \times h$ pixels, where each pixel contains a colour from a semiring $\mathcal{C}$ of possible colours. The image can be written as a function*

$$\{0, 1, 2, \ldots, w - 1\} \times \{0, 1, 2, \ldots, h - 1\} \to \mathcal{C}.$$

In practice, the range is normally some predetermined interval such as $[0, 1]$ or a set $\{0, 1\}$ where 0 is interpreted as black and 1 as white, while the intermediate numbers represent the shades of grey. Images with $\mathcal{C} = \{0, 1\}$ are called bilevel images and images with $\mathcal{C} = [0, 1]$ are greyscale. For coloured images, the colour set is typically $[0, 1]^3$, where the colour is represented as a vector with three numbers representing the intensities of three colour components, for example, red, green and blue for the RGB[1] representation. The images with different colour sets are shown in figure 1.1.

Because we use digital images, the colour is commonly quantized to 8 bits per pixel ($3 \cdot 8$ bits for colour images), which gives us 255 possible values for intensity. In this thesis, to make the theory simpler, we set $\mathcal{C} = \mathbb{R}$ for greyscale images and $\mathcal{C} = \mathbb{R}^3$ for colour images. Also, in the context of this thesis, we will only be referring to the images whose dimensions are

$$w = h = 2^k \text{ for some } k \in \mathbb{Z}_+.$$

This simplification serves as a more straightforward way to explain how weighted automata work with images.

At the end of this section, we talk about what changes we have to implement to be able to work with images of different resolutions.

[1] RGB colour representation [6]

Figure 1.1: Bilevel, greyscale and colour images.

1.1.1 Addressing Pixels

Definition 3. *Let Σ be a non-empty set of letters. Then*

- *a **word** w is a finite (or empty) sequence of letters $s \in \Sigma$, empty word is denoted as ϵ,*
- *the set of all words over Σ is denoted as Σ^*,*
- *the set of all non-empty words is denoted as Σ^+,*
- *the set of all words of length n is denoted as Σ^n,*
- *$|w|$ is the length of the word w.*

Typically, pixels in an image are addressed by their x- and y-coordinates. If we translated those coordinates into words using the definition 3, we will get an addressing scheme where each word would be of length 2, with the first letter being the x coordinate and the second the y coordinate. Unfortunately, the alphabet of those words would depend on the size of the image. However, to be able to use automata on images, the alphabet needs to have a fixed size. Therefore, we present a different addressing scheme.

The basic idea is to represent the pixels using words over the four-letter alphabet $\Sigma = \{0, 1, 2, 3\}$, where each pixel of a $2^k \times 2^k$ image is addressed by a word of length k. The addressing works as follows: First, the image is divided into four quadrants. Each quadrant gets assigned an address using the letters of our alphabet Σ, as shown in figure 1.2. Each of those quadrants is viewed as an image of size $2^{k-1} \times 2^{k-1}$. They are subsequently divided into smaller quadrants, whose addresses are now two-letter words. This process is repeated until we reach depth k, where each image is of size 1×1, i.e. they are single pixels.

In another words, the four quadrants of a sub-square with address w are addressed by words $w0$, $w1$, $w2$ and $w3$, where wa for $a \in \Sigma$ is a concatenation of word w and the letter a. The whole picture is addressed by the empty word ϵ.

Example 1. Figure 1.3 shows an image of size $2^3 \times 2^3$ where each pixel is addressed by a word of length 3. The second illustration highlights a pixels with address 0312.

Now we can define a $2^k \times 2^k$ image as a function

$$f_k : \Sigma^k \to \mathcal{C}$$

that assigns a colour $c \in \mathcal{C}$ to a pixel addressed by a word of length k.

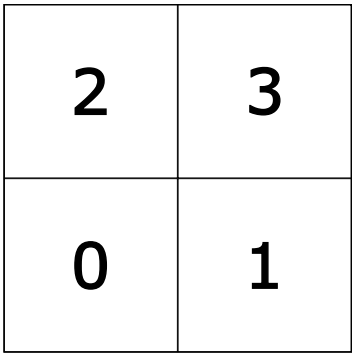


Figure 1.2: The addresses of quadrants.

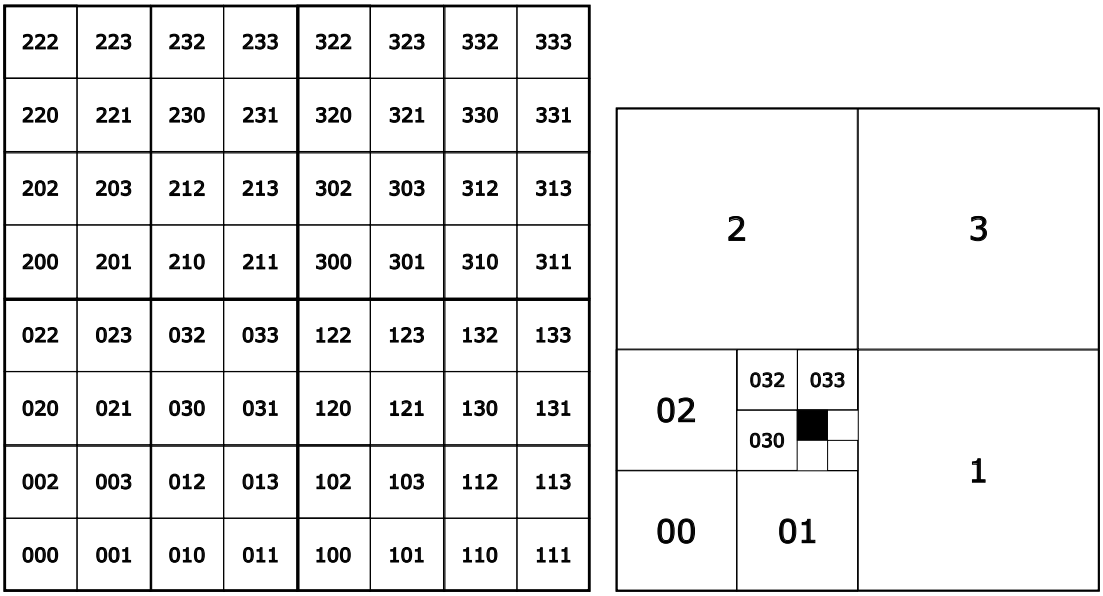


Figure 1.3: The addresses of pixels in resolution $2^3 \times 2^3$ and the pixel with address 0312, respectively.

Images of different sizes

As we have shown, the division of an image of size $2^k \times 2^k$ into four equal quadrants works quite naturally. However, most real-world images are rectangular and do not have dimensions aligned to the powers of 2. Working with images of arbitrary dimensions would entail working with sub-rectangles rather than sub-squares, which would be very complicated. Instead, for other resolution sizes, the picture can be enclosed with auxiliary black pixels to form a picture with resolution $2^k \times 2^k$ for some $k \geq 0$.

Another option would be using a different alphabet. [7] shows an addressing scheme with $\Sigma = \{0, 1\}$, which is able to deal with arbitrary picture sizes.

1.2 Multi-resolution Images

Definition 4. *A **multi-resolution image** is a function*

$$f : \Sigma^* \to \mathcal{C}$$

that assigns a colour to each pixel in resolution $2^k \times 2^k$ for all $k \geq 0$.

If we restrict f to words of length k, such as $f_k = f_{|_{\Sigma^k}}$, then we get a $2^k \times 2^k$ image. Therefore, we can view the multi-resolution image as a sequence $f_0, f_1, f_2, \ldots$ of finite-resolution images.

Another way to represent a multi-resolution image f with our addressing scheme is by a **quad-tree** (a tree data structure where each node has four children). The quad-tree representing a multi-resolution image is constructed as follows: The root of the quad-tree is labelled as $f(\epsilon)$ and its four sons are labelled left to right by the letters from Σ as $f(0)$, $f(1)$, $f(2)$ and $f(3)$ for each quadrant (see figure 1.4). Every word $w \in \Sigma$ is then an address of a unique node of the quad-tree at depth $|w|$. The children of the node addressed by w are $f(w0)$, $f(w1)$,$f(w2)$ and $f(w3)$. Therefore, each level k of the quad-tree represents the finite-resolution image of size $2^k \times 2^k$.

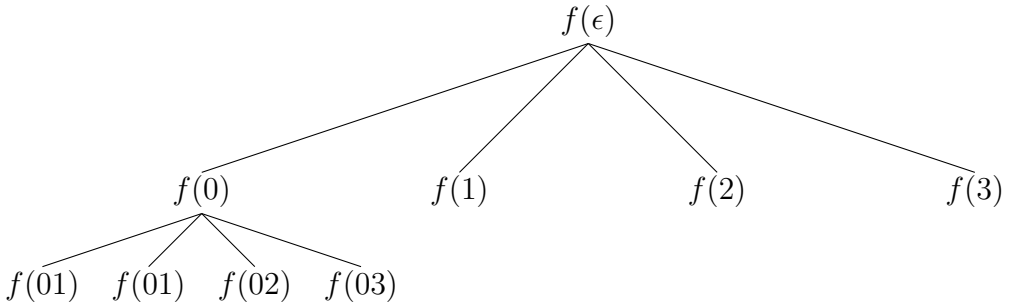


Figure 1.4: A quad-tree representing a multi-resolution image f.

We would like to point out that the definition 4 of a multi-resolution image does not require any similarities between the finite-resolution images it contains. But for our purposes, we define an average preserving multi-resolution image so that each finite-resolution image f_k is an approximation of the same infinitely sharp image.

Definition 5. *We say that the multi-resolution image f is **average preserving**, or **ap**, if for all $w \in \Sigma^*$ holds*

$$f(w) = \frac{1}{4}\left(f(w0) + f(w1) + f(w2) + f(w3)\right).$$

The ap characteristic of the image means that for a given image, $f(w)$ is the average colour of its children. In this case, one can easily move from higher to lower resolution by simply computing the averages of the intensities inside each sub-square.

From another point of view, f_{k-1} is an interpolation of the next sharper image f_k and therefore, the sequence of finite-resolution images $f_0, f_1, f_2, \ldots$ is a sequence of sharper and sharper approximations of some multi-resolution image f. Figure 1.5 shows finite-resolution approximations $f_2, f_3, f_4, f_5, f_6, f_7$ of an ap multi-resolution image f.

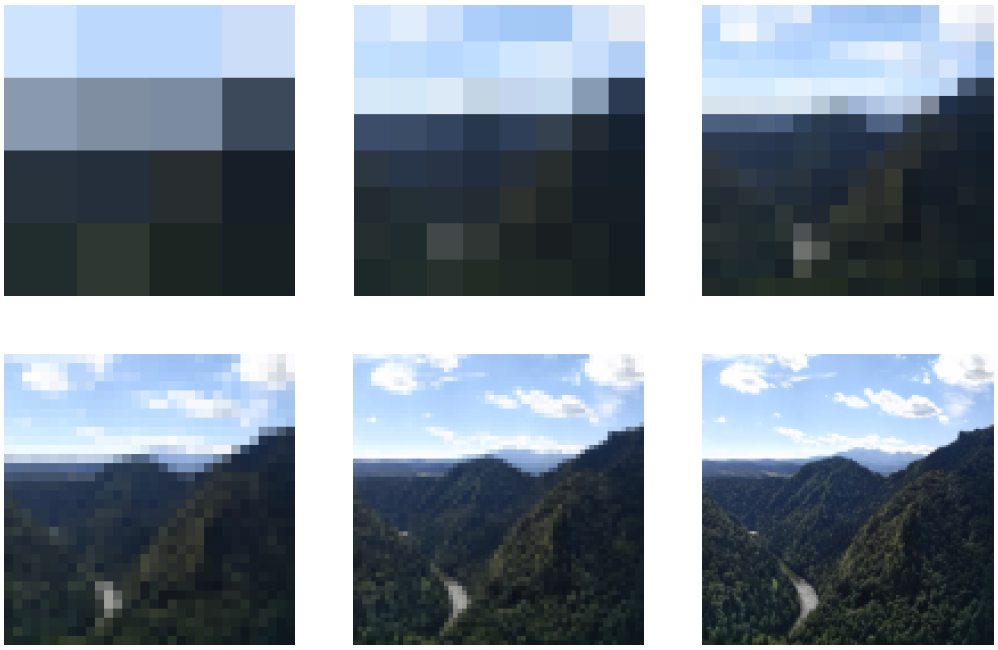


Figure 1.5: Finite resolution images $f_2, f_3, f_4, f_5, f_6, f_7$.

The set

$$\mathcal{F} = \{f \mid f : \Sigma^* \to \mathbb{R}\}$$

of all multi-resolution images is a linear space where the operation of addition and multiplication by real numbers are defined pixel-wise as follows:

$$(f + g)(w) = f(w) + g(w)\ \forall w \in \Sigma^*$$
$$(r \cdot f)(w) = r \cdot f(w)\ \forall w \in \Sigma^*, r \in \mathcal{C}$$

The set of the average preserving multi-resolution images is a linear subspace of $\mathcal{F}$.

The images with $2^k \times 2^k$ resolution also form a linear space that is isomorphic to $\mathcal{C}^{4^k}$. Thanks to this, we can use standard algorithms of linear algebra. For example, we can express a finite-resolution image f_k as a linear combination of finite-resolution images $\psi_1, \ldots, \psi_n$ as

$$f_k = c_1 \cdot \psi_1 + c_2 \cdot \psi_2 + \ldots + c_n \cdot \psi_n$$

if such $c_1, \ldots c_n \in \mathcal{C}$ exist.

These algebraic properties of our addressing scheme will be used in the encoding and decoding algorithms described in chapter 3.

2. Weighted Finite Automata

In this section, we will introduce weighted finite automata (WFA) as a generalization of non-deterministic finite automata and mention some of their elementary properties. Then we will show how we can apply the WFA in image representation. For this, we will use the addressing scheme with Σ as the four-letter alphabet $\Sigma = \{0, 1, 2, 3\}$ we introduced in the chapter 1.

Definition 6. *A **non-deterministic finite automaton** (NFA) is specified by*

- *a finite set of states Q,*
- *a finite alphabet Σ,*
- *a transition function $\delta : Q \times \Sigma \times Q \to \{0, 1\}$,*
- *a set of initial states $I \subseteq Q$,*
- *a set of final states $F \subseteq Q$.*

Note, that in most literature, the transition function is defined as

$$\delta' : Q \times \Sigma \to \mathcal{P}(Q),$$

where $\mathcal{P}(Q)$ denotes the power set of Q. We changed that definition because it will then be easier to generalize it to WFA. The relationship between our definition and the common definition is as follows:

$$\delta'(q, a) = \{p \mid \delta(q, a, p) = 1\}.$$

The weighted finite automata, as we already mentioned, are generalized NFA. Namely, NFA is only a special case of WFA over the boolean semiring. The generalization lies in the transition function. Whereas the transition function of NFA returns only 0 or 1, which indicates whether the transition is in the automaton or not, the transition function of WFA returns an element of a semiring, which is called the weight of a transition.

Remark. In this chapter, R will denote an arbitrary semiring.

Definition 7. *A **weighted finite automaton** over a semiring R is specified by:*

- *a finite set of states Q, where $|Q| = n$,*
- *a finite alphabet Σ,*
- *a weight function $\delta : Q \times \Sigma \times Q \to R$,*
- *an initial distribution $\alpha : Q \to R$,*
- *a final distribution $\beta : Q \to R$.*

The WFA then defines a function $f : \Sigma^* \to R$ *as follows:*

$$f(a_1, a_2, \ldots, a_k) = \sum_{q_0,\ldots q_k \in Q} \alpha(q_0) \cdot \delta(q_0, a_1, q_1) \cdot \delta(q_1, a_2, q_2) \cdot \ldots \cdot \delta(q_{k-1}, a_k, q_k) \cdot \beta(q_k)$$

for all $a_1 a_2 \ldots a_k \in \Sigma^*$.

If $\alpha(q) \neq 0$, we say that the state q is an initial state; if $\beta(q) \neq 0$ we say that q is a final state.

Definition 8. *Let* A *be a WFA. We say that* $(q_0, a_1, q_1, a_2, \ldots a_k, q_k)$ *is a* ***path*** *of length* k *in* A *for* $q_0, \ldots q_k \in Q$ *over a word* $a_1, a_2, \ldots, a_k \in \Sigma^k$.

The ***weight*** *of the path* $(q_0, a_1, q_1, a_2, \ldots a_k, q_k)$ *in* A *is the product of*

- *the initial distribution of* q_0,
- *the weights of transitions from* q_i *to* q_{i+1} *with label* a_{k+1} *for* $i = 0 \ldots k-1$,
- *the final distribution of* q_k.

Let word $w = a_1, a_2, \ldots, a_k \in \Sigma^*$. The function f defined by a WFA can be read as follows: $f(w)$ is obtained by taking all paths in the automaton, whose labels form the word w. Then, $f(w)$ is the sum of the weights of all such paths.

A more convenient representation of weight function δ can be written as

$$\forall a \in \Sigma,\ \forall p, q \in Q :\ (A_a)_{p,q} = \delta(p, a, q)$$

where $A_a \in R^{n \times n}$ is a weight matrix of letter a. If the element $(A_a)_{p,q} = 0$, we say that there is no transition from state p to state q with label a. The initial distribution can be described by a row vector $I \in R^{1 \times n}$, where $I_q = \alpha(q)$ for $q \in Q$; for simplicity of notation, we assume that $Q = \{0, 1, \ldots, n-1\}$. Analogically we can describe the final distribution as a column vector $F \in R^{n \times 1}$ where $F_q = \beta(q)$ for $q \in Q$.

We are aware that the notation of initial and final distribution vectors resembles matrix notation, where capital letters denote matrices. Nevertheless, we used this notation to maintain consistency with other articles about this topic.

Let $w = a_1, a_2, \ldots, a_k \in \Sigma^*$. Then we denote matrix A_w as

$$A_w = A_{a_1} \cdot A_{a_2} \cdot \ldots \cdot A_{a_k}.$$

Lemma 1. *Let be* A_w *for word* $w \in \Sigma^*$. *Then for all paths starting at state* $q_i \in Q$ *and ending in* $q_j \in Q$ *over a word* w *is*

$$\sum_{\substack{q_0,\ldots q_k \in Q \\ q_0 = q_i, q_k = q_j}} \delta(q_0, a_1, q_1) \delta(q_1, a_2, q_2) \ldots \delta(q_{k-1}, a_k, q_k) = (A_w)_{q_i, q_j}.$$

Proof. For word v with length 1 the lemma holds. Now, let us assume, that lemma holds for words with length $k-1$.

Then for word $w = a_1, a_2, \ldots, a_k$ of length k:

$$\begin{aligned}
(A_w)_{q_i,q_j} &= (A_{a_1} \cdot A_{a_2} \cdot \ldots \cdot A_{a_k})_{q_i,q_j} \\
&= \sum_{q_l \in Q} (A_{a_1})_{q_i,q_l} \cdot \sum_{\substack{q_1,\ldots q_k \in Q \\ q_1=q_l, q_k=q_j}} (A_{a_2} \cdot \ldots \cdot A_{a_k})_{q_l,q_j} \\
&= \sum_{q_l \in Q} (A_{a_1})_{q_i,q_l} \cdot \sum_{\substack{q_1,\ldots q_k \in Q \\ q_1=q_l, q_k=q_j}} \delta(q_1, a_2, q_2) \ldots \delta(q_{k-1}, a_k, q_k) \\
&= \sum_{q_l \in Q} \sum_{\substack{q_1,\ldots q_k \in Q \\ q_1=q_l, q_k=q_j}} (A_{a_1})_{q_i,q_l} \cdot \delta(q_1, a_2, q_2) \ldots \delta(q_{k-1}, a_k, q_k) \\
&= \sum_{q_l \in Q} \sum_{\substack{q_1,\ldots q_k \in Q \\ q_1=q_l, q_k=q_j}} \delta(q_i, a_1, q_1) \cdot \delta(q_1, a_2, q_2) \ldots \delta(q_{k-1}, a_k, q_k) \\
&= \sum_{\substack{q_0,\ldots q_k \in Q \\ q_0=q_i, q_k=q_j}} \delta(q_0, a_1, q_1) \cdot \delta(q_1, a_2, q_2) \ldots \delta(q_{k-1}, a_k, q_k)
\end{aligned}$$

In the third equality, we used the induction hypothesis. □

Theorem 2. *WFA A with transition matrices A_a for all $a \in \Sigma$, initial distribution $I \in R^{1\times n}$ and final distribution $F \in R^{n\times 1}$, defines function $f : \Sigma^* \to R$ as follows:*

$$f(w) = IA_wF$$

for all $w = a_1, a_2, \ldots, a_k \in \Sigma^*$.

Proof. Let $a_1, a_2, \ldots, a_k \in \Sigma^*$. Then

$$\begin{aligned}
f(w) &= \sum_{q_0,\ldots q_k \in Q} \alpha(q_0)\delta(q_0, a_1, q_1)\delta(q_1, a_2, q_2) \ldots \delta(q_{k-1}, a_k, q_k)\beta(q_k) \\
&= \sum_{q_i,q_j \in Q} \alpha(q_i)(A_w)_{q_i,q_j}\beta(q_j) \\
&= \sum_{q_i,q_j \in Q} I_{q_i}(A_w)_{q_i,q_j}F_{q_j} \\
&= \sum_{q_i \in Q} \sum_{q_j \in Q} I_{q_i}(A_w)_{q_i,q_j}F_{q_j} \\
&= IA_wF
\end{aligned}$$

In the second equality, we used the lemma 1. □

Definition 9. *A WFA A is **average preserving** (ap-WFA) if the following applies:*

$$(A_0 + A_1 + A_2 + A_3) \cdot F = 4F$$

Note, that if the WFA is average preserving, then the final distribution F is an eigenvector of matrix $A_0 + A_1 + A_2 + A_3$ corresponding to eigenvalue 4.

Definition 10. *Let f be a function defined by a WFA A. We say f is **average preserving** if for all $w \in \Sigma^*$ holds*

$$f(w) = \frac{1}{4}\left(f(w0) + f(w1) + f(w2) + f(w3)\right).$$

Theorem 3. *Let A be a WFA. If A is average preserving, then the function f defined by the WFA A is average preserving.*

Proof. Let $w \in \Sigma^*$ be arbitrary. From theorem 2 of the function computed by WFA we get

$$f(w) = IA_wF = I \cdot A_{a_1}A_{a_2}\dots A_{a_k} \cdot F$$

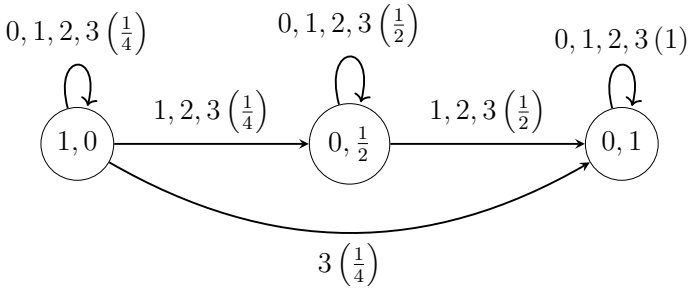
Then

$$\begin{aligned}\sum_{a\in\Sigma} f(wa) &= \sum_{a\in\Sigma} IA_wA_aF \\ &= IA_w\left(\sum_{a\in\Sigma} A_a\right)F \\ &= 4 \cdot IA_wF \\ &= 4 \cdot f(w)\end{aligned}$$

In the third equality, we used the definition of average preserving WFA. This proves, that f satisfies the definition of average preserving function for every word $w \in \Sigma^*$, which means that f is average preserving. □

Commonly, the WFA is drawn as a labelled, weighted directed graph, where the initial and final distribution values are marked inside the nodes. The transition weights are drawn in parentheses after the label of the transition. Transitions with weight 0 are usually omitted.

Example 2. Consider the WFA A over $R = \mathbb{R}$ below



For example, the value $f(03)$ is the sum of all twenty-seven paths whose label reads 03, however, for clarity we will only write weights of paths starting in q_0, because it is the only initial state:

$$f(03) = 1 \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot 0 + 1 \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{2} + 1 \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot 1 = 0 + \frac{1}{32} + \frac{1}{16} = \frac{3}{32}$$

The matrix representation of A is:

$$I = \begin{pmatrix}1 & 0 & 0\end{pmatrix} \qquad A_0 = \begin{pmatrix}\frac{1}{4} & 0 & 0\\ 0 & \frac{1}{2} & 0\\ 0 & 0 & 1\end{pmatrix} \qquad A_1 = \begin{pmatrix}\frac{1}{4} & \frac{1}{4} & 0\\ 0 & \frac{1}{2} & \frac{1}{2}\\ 0 & 0 & 1\end{pmatrix}$$

$$F = \begin{pmatrix}0\\ \frac{1}{2}\\ 1\end{pmatrix} \qquad A_2 = \begin{pmatrix}\frac{1}{4} & \frac{1}{4} & 0\\ 0 & \frac{1}{2} & \frac{1}{2}\\ 0 & 0 & 1\end{pmatrix} \qquad A_3 = \begin{pmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4}\\ 0 & \frac{1}{2} & \frac{1}{2}\\ 0 & 0 & 1\end{pmatrix}$$

The value of $f(03)$ is calculated, using matrix representation, as follows:

$$f(03) = IA_0A_3F = \frac{3}{32}.$$

2.1 Multiplying Two WFA

Given an n-state WFA A and an m-state WFA B that define multi-resolution functions f and g respectively, it is trivial to get $f_A(w) \cdot f_B(w)$ for some $w \in \Sigma^*$. First, we will compute the value for $f(w)$, then for $g(w)$, and then we multiply the those values.

However, what do we do if we want to use only one automaton to compute $f_A(w) \cdot f_B(w)$? For this, we will define the multiplication of two WFA because combining two WFA by multiplication is a powerful tool used in a variety of applications to create complex WFA from simpler ones. Also, we will use it in our encoding algorithm in section 3.2.1.

The process of the multiplication of weighted automata is a generalization of the standard multiplication algorithm for finite automata.

First, let us introduce a notation for this section:

- Let A be an n-state WFA A with set of states Q_A, alphabet Σ_A, transition matrices A_a for all $a \in \Sigma_A$, initial distribution I_A and final distribution F_A that defines multi-resolution function f_A.
- Then let B an m-state WFA with set of states Q_B, alphabet Σ_B, transition matrices B_a for all $a \in \Sigma_B$, initial distribution I_B and final distribution F_B that defines multi-resolution function f_B.

Then we create an nm-state WFA C as follows:

- a set of states $Q = Q_A \times Q_B = \{(p_i, q_j) \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}\}$,
- an alphabet $\Sigma = \Sigma_A \cap \Sigma_B$,
- the transition matrices $C_a \in R^{nm \times nm}$ for all $a \in \Sigma$, where

  $$(C_a)_{(p_i,q_j),(p_k,q_l)} = (A_a)_{p_i,p_k} \cdot (B_a)_{q_j,q_l}$$

  for all $p_i, p_k \in Q_A$ and $q_j, q_l \in Q_B$,
- an initial distribution $I \in R^{1 \times nm}$ where

  $$I_{(p_i,q_j)} = (I_A)_{p_i} \cdot (I_B)_{q_j}$$

  for all $(p_i, q_j) \in Q$,
- an final distribution $F \in R^{nm \times 1}$ where

  $$F_{(p_i,q_j)} = (F_A)_{p_i} \cdot (F_B)_{q_j}$$

  for all $(p_i, q_j) \in Q$.

Then the WFA C defines a function $f_C : \Sigma^* \to R$ as follows:

$$f_C(w) = (fg)(w),$$

and from the theorem 2 we get that

$$f_C(w) = IC_wF$$

for every word $w \in \Sigma^*$.

Theorem 4. *The WFA C created above defines a function f_C as follows:*

$$f_C(w) = f_A(w) \cdot f_B(w)$$

for every word $w \in \Sigma^*$.

Proof. Let $w = a_1 \dots a_k \in \Sigma^*$. Then
$f_A(w) \cdot f_B(w) =$

$$\begin{aligned}
&= \left(\sum_{p_0,\dots p_k \in Q_A} (I_A)_{p_0} (A_{a_1})_{p_0,p_1} \cdot \dots \cdot (A_{a_k})_{p_{k-1},p_k} (F_A)_{p_k} \right) \cdot \\
&\qquad \cdot \left(\sum_{q_0,\dots q_k \in Q_B} (I_B)_{q_0} (B_{a_1})_{q_0,q_1} \cdot \dots \cdot (B_{a_k})_{q_{k-1},q_k} (F_B)_{q_k} \right) \\
&= \sum_{p_0,\dots p_k \in Q_A} \sum_{q_0,\dots q_k \in Q_B} (I_A)_{p_0} (A_{a_1})_{p_0,p_1} \cdot \dots \cdot (A_{a_k})_{p_{k-1},p_k} (F_A)_{p_k} \cdot \\
&\qquad \cdot (I_B)_{q_0} (B_{a_1})_{q_0,q_1} \cdot \dots \cdot (B_{a_k})_{q_{k-1},q_k} (F_B)_{q_k} \\
&= \sum_{p_0,\dots p_k \in Q_A} \sum_{q_0,\dots q_k \in Q_B} (I_A)_{p_0} (I_B)_{q_0} (A_{a_1})_{p_0,p_1} (B_{a_1})_{q_0,q_1} \cdot \dots \\
&\qquad \cdot \dots \cdot (A_{a_k})_{p_{k-1},p_k} (B_{a_k})_{q_{k-1},q_k} (F_A)_{p_k} (F_B)_{q_k} \\
&= \sum_{(p_0,q_0)\dots(p_k,q_k) \in Q} I_{(p_0,q_0)} (C_{a_1})_{(p_0,q_0),(p_1,q_1)} \dots (C_{a_k})_{(p_{k-1},q_{k-1}),(p_k,q_k)} F_{(p_k,q_k)} \\
&= \sum_{(p_0,q_0),(p_k,q_k) \in Q} I_{(p_0,q_0)} (C_w)_{(p_0,q_0),(p_k,q_k)} F_{(p_k,q_k)} \\
&= IC_wF = f_C(w)
\end{aligned}$$

We used lemma 1 in the second last equality. □

Note that the construction of initial and final distribution is analogous to the construction of initial and final states when multiplying finite automata. When we multiply two finite automata, the initial state of the resulting automaton is the state that combines both initial states of the original automata. In other words, the resulting finite automaton has the initial state where the multiplication of the initial distribution of the original states is non-zero. Analogously, this applies to the final distribution of WFA multiplication.

Another motivation behind describing the WFA multiplication was to be able to create WFA calculating polynomials of several variables. It was shown in [8] that each real polynomial $P(x)$ with non-negative coefficients of degree d defined in $[0, 1]$ can be computed by a $d + 1$ state WFA and with defined multiplication there exists a WFA for every polynomial $P(x_1, \dots, x_n)$ with non-negative coefficient and defined in $[0, 1]$.

The polynomials will be used in the encoding algorithm in section 3.2.1.

Example 3. In the figure 2.1 we can see automata generating the linear polynomials constructed by algorithm from [8]. On the right, we see automaton xy built by the two automata using the multiplication operation.

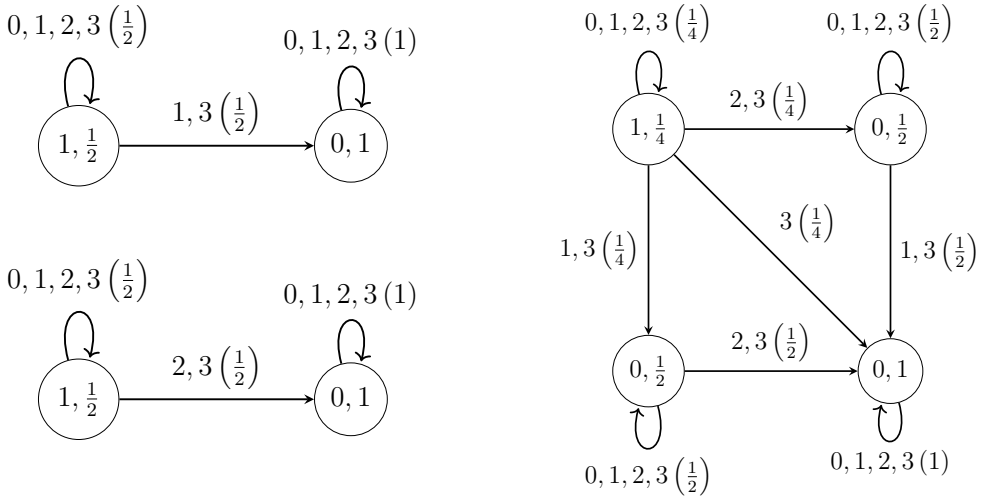


Figure 2.1: WFA defining poylnomial functions x (above), y (below) and xy (right).

2.2 Representing Images With WFA

In this section, we will use the addressing scheme from chapter 1 to represent images with WFA.

A function $f : \Sigma^* \to R$ defined by a WFA defines a multi-resolution image from section 1.2, such that the colour of the pixel with address $w \in \Sigma^*$ is $f(w)$. From theorem 3 it holds, that if the WFA is ap than also the multi-resolution image is ap.

As we mentioned in the section 1.1, the colours in digital images are typically quantized to 8 bits per pixel. Therefore, in this section, we will be using the weighted automata over an interval $[0, 1]$ or $[0, 1]^3$ instead the semiring R because they can be easily mapped to the quantized 8 bits per pixel for greyscale and 24 bits for coloured images respectively.

Example 4. If we take the WFA from example 2, then the multi-resolution image computed by the WFA is shown in figure 2.2.

Coloured images are composed from three different greyscale colour layers. For example, the most widely used RGB format uses red, green, and blue colour components. By using different initial distributions I_R, I_G, I_B for each colour layer, we can define colour images with only one WFA.

Example 5. Consider the following WFA:

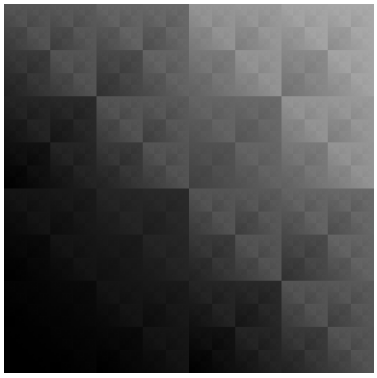
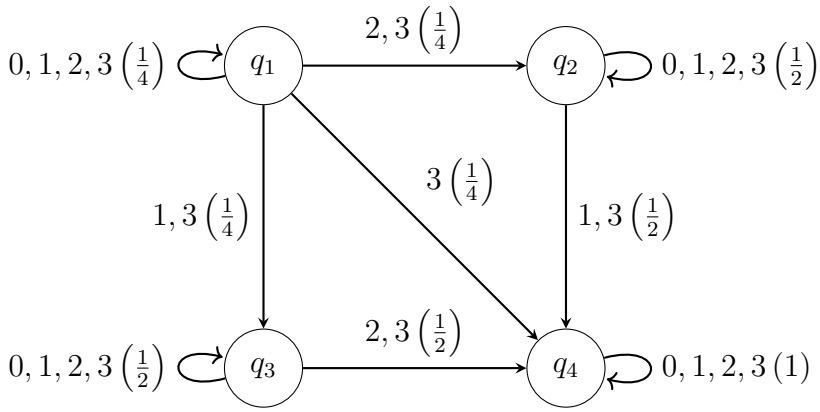


Figure 2.2: An image generated by the WFA from example 2.



with three initial distributions

$$I_R = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$$
$$I_G = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}$$
$$I_B = \begin{pmatrix} -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & 1 \end{pmatrix}$$

and final distribution $F^T = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 \end{pmatrix}$.

With each initial distribution, the WFA defines a different greyscale image. If we assign the individual greyscale images to the colour components (first distribution to the red, second to green and third to blue), we get the colour image shown in figure 2.3.

If we change the initial distribution to:

$$I_R = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$$
$$I_G = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}$$
$$I_B = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 1 \end{pmatrix}$$

We get the colour image shown in figure 2.4.

In the section 2.1 we discussed multiplication of WFA as well as using WFA to represent polynomial functions with multiple variables. In figure 2.5 you can see the images defined by the WFA from example 3.

As we mentioned in section 2.1, multiplication of two automata can be used when creating complex WFA from simpler automata. In figure 2.7 you can see

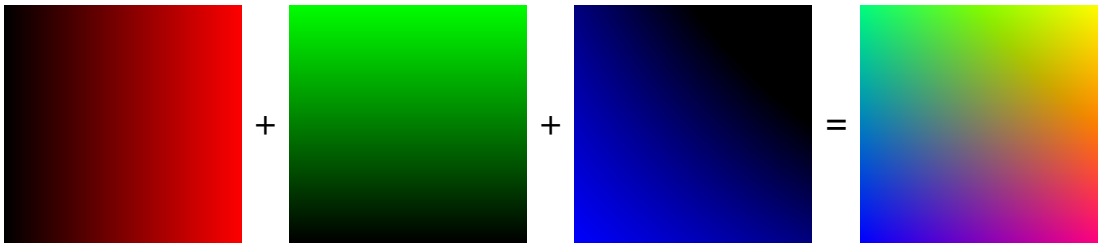


Figure 2.3: Three colour components defined by different initial distributions and their resulting colour image (defined by WFA from example 5).

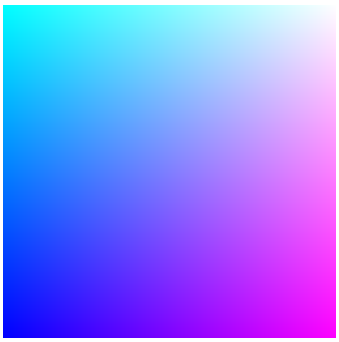


Figure 2.4: Coloured image defined by WFA from example 5 with different initial distributions.

the product of the 4-state automaton from example 5 and a one-state automaton generating the Sierpinski triangle (see figure 2.6) creating a four state WFA.

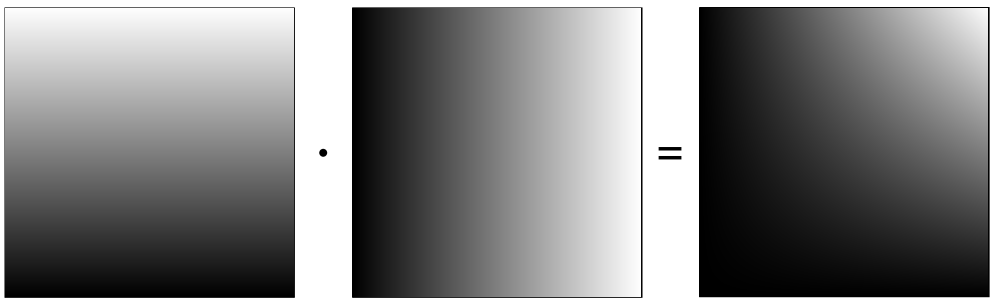


Figure 2.5: Automaton computing xy.

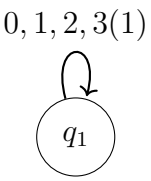


Figure 2.6: WFA generating the Sierpinki triangle.

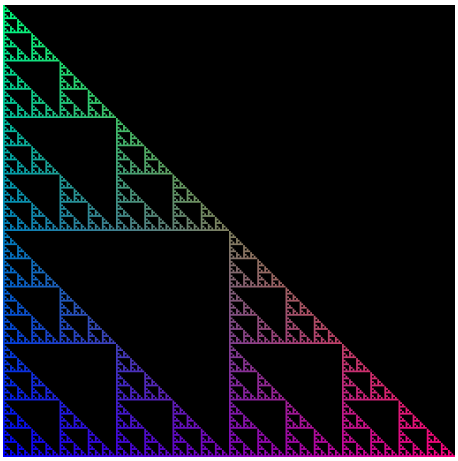


Figure 2.7: Image generated by a four-state WFA.

3. Image Generation And the Encoding Algorithm

In this chapter we describe how we can generate images from WFA. Also, we explain how an image can be encoded into a WFA.

3.1 Image Generation

Assume we are given a WFA A, and we want to generate the image it represents with some specific finite-resolution $2^k \times 2^k$. We only give here the algorithm. If you would like to know more, such as the time complexity, see [3].

Decoding Algorithm

1. For all non-empty words $w \in \Sigma^{\leq \lfloor \frac{k}{2} \rfloor}$ compute the product IA_w. For all words $v \in \Sigma^{\leq \lceil \frac{k}{2} \rceil}$ compute the product A_vF. The calculations will proceed gradually over the increasing length of words w and v.
2. Compute the product of all possible pairs $(IA_w)(A_vF)$ for all $w \in \Sigma^{\lfloor \frac{k}{2} \rfloor}$ and $v \in \Sigma^{\lceil \frac{k}{2} \rceil}$.

The goal of the algorithm is to compute IA_wF for all $w \in \Sigma^k$. In the first step, we divide the computation into two halves. We need to proceed gradually with increasing length of the words, but we are only interested in values for the longest words. The second step only combines the intermediate results. This approach is more efficient than computing IA_wF directly.

3.2 Encoding Images Into WFA

In section 2.2 we explained how WFA can represent multi-resolution images. We will now describe how we can create a WFA that represents a given multi-resolution image. For simplicity, we will consider only greyscale images. This restriction is not strict, since each colour component of a colour image can be treated as a stand-alone greyscale image. Also, we will be working only with ap multi-resolution images, because we need the ability to compute images with lower resolutions from images with higher resolutions.

The algorithm will use self-similarities of the input image. Consider a multi-resolution image f with a colour set $\mathcal{C}$. If a sub-square of f is similar to some other sub-squares of f in terms of multiplication by elements of $\mathcal{C}$, then the automaton will express this sub-square as a linear combination of the similar sub-squares.

Notation

Before we write down the algorithm, we need to introduce a notation which will be then used in the algorithm. The notation was taken from the article [3].

Let us consider a set of images generated by a WFA A, where each image is obtained from A by changing the initial distribution so that the initial distribution

value of state i is 1, and all other states have initial distribution 0. In another words, let us denote a multi-resolution image ψ_i for every $i \in Q_A$ as follows:

$$\psi_i(w) = (A_w F)_i,\ w \in \Sigma^*.$$

We say that a multi-resolution image ψ_i is an image of state i.

This notation gives us a recursive relationship between the multi-resolution images ψ_i:

- $\psi_i(\epsilon) = F_i$, which means that the final distribution consists of average intensities of the state images.
- For each $a \in \Sigma$ and $w \in \Sigma^*$

$$\begin{aligned}(\psi_i)_a(w) = \psi_i(aw) = (A_{aw}F)_i = (A_a A_w F)_i \\ = (A_a)_{i,1}(A_w F)_1 + (A_a)_{i,2}(A_w F)_2 + \cdots + (A_a)_{i,n}(A_w F)_n \\ = r_1\psi_1(w) + r_2\psi_2(w) + \ldots + r_n\psi_n(w)\end{aligned}$$

  where $r_j = (A_a)_{ij}$ is the weight of the transition from state i to state j. In other words, the i-th row of transition matrix A_a provides the coefficients $r_1, \ldots, r_n$ for the linear combination of the state images which gives us $(\psi_i)_a$.

  Since the coefficients r_j are independent of the word w, we can write

$$(\psi_i)_a = r_1\psi_1 + r_2\psi_2 + \ldots + r_n\psi_n$$

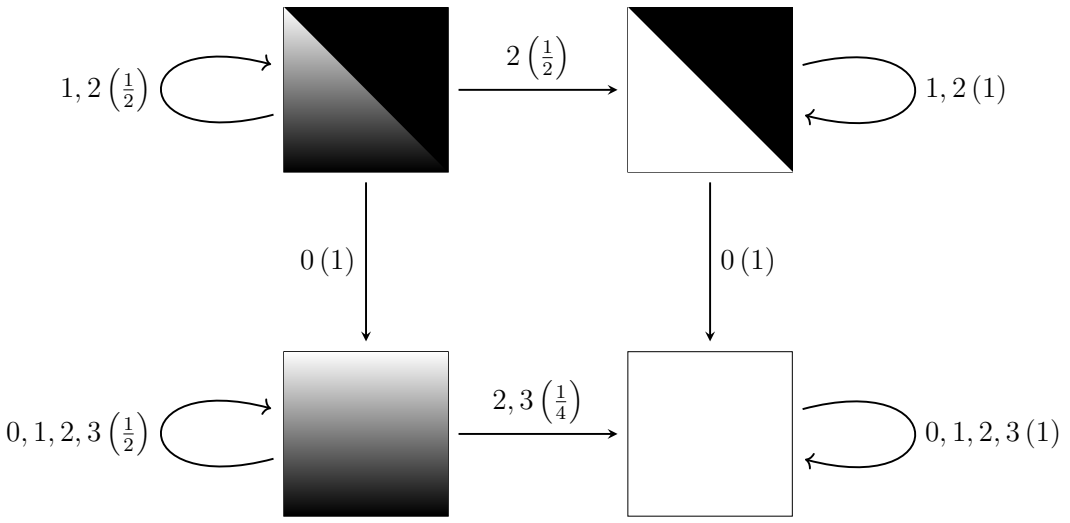
  With this interpretation, we can express the quadrant a of the image ψ_i as the linear combination of state images $\psi_1, \ldots, \psi_n$ with coefficients of the i-th row of the transition matrix A_a.

- The multi-resolution image f defined by the WFA A can be expressed as

$$f = I_1\psi_1 + I_2\psi_2 + \ldots + I_n\psi_n.$$

  Specifically, the initial distribution I gives the coefficients to express the multi-resolution image f as a linear combination of state images $\psi_1, \ldots, \psi_n$.

Example 6. Let us consider following the WFA with the images ψ_i shown inside the nodes.



Here we use $[0, 1]$ as the colour set where 0 is black and 1 is white. Figure 3.1 shows the linear expressions denoted by $(\psi_1)_1$ and $(\psi_1)_2$. Note, that the transitions in the fourth state (down-right) of the automaton indicate that all quadrants of ψ_4 are the same as the image ψ_4.

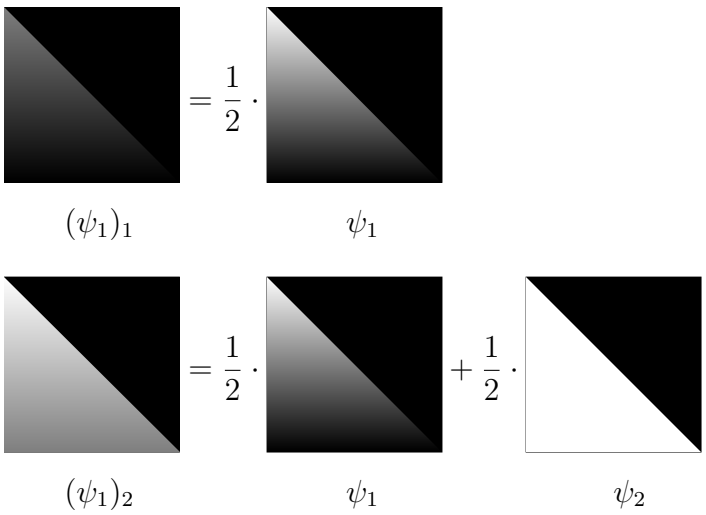


Figure 3.1: Linear combination of images expressing $(\psi_1)_1$ and $(\psi_1)_2$ from example 6.

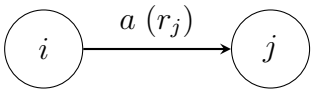
Encoding Algorithm

We will show how we can find a WFA that represents a given multi-resolution image. The outline of the encoding algorithm is cited from [3]:

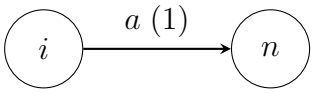
Input: multi-resolution image ψ

Variables: n : number of states in the automaton so far
ψ_j : image of state j, $1 \leq j \leq n$
i : first non-processed state

1. $n \leftarrow 1$, $i \leftarrow 1$, $\psi_1 \leftarrow \psi$
2. For all quadrants $a = 0, 1, 2, 3$ do:
   a) If $\exists r_1, \ldots r_n \in \mathbb{R}$ such that $(\psi_i)_a = r_1\psi_1 + r_2\psi_2 + \ldots + r_n\psi_n$ then add the transitions



   for all $j = 1, 2, \ldots n$.
   b) Else create a new state: Set $n \leftarrow n + 1$, $\psi_n \leftarrow (\psi_i)_a$ and add the transition

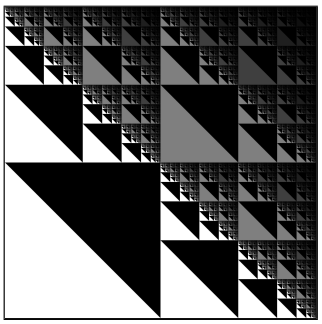


3. $i \leftarrow i + 1$. If $i \leq n$ go to 2.

4. Initial distribution: $I_1 = 1$, $I_i = 0$ for $i = 2, 3 \ldots, n$
   Final distribution $F_i = \psi_i(\epsilon)$ for $i = 1, 2 \ldots, n$

For better understanding, we will demonstrate how the algorithm works in the following example.

Example 7. The input for the algorithm will be the following multi-resolution image:



Let us find a WFA that represents this image. We assign state q_1 to the whole image ψ, therefore $\psi_1 = \psi$. We then process the four quadrants of ψ_1 starting in the bottom-left quadrant $(\psi_1)_0$. We can see that this quadrant contains an image that cannot be expressed as a linear combination of previously processed states, therefore we create a new state q_2 for this quadrant. The quadrants $(\psi_1)_1$ and $(\psi_1)_2$ are the same as the input image, so we create new transitions from the state q_1 to q_1 with labels 1 and 2, both weights being 1. Next, we process the quadrant $(\psi_1)_3$. The quadrant is the same as the input image, but with only half the intensity. Therefore, we create a new transition from q_1 to q_1 labelled 3 with weight $\frac{1}{2}$.

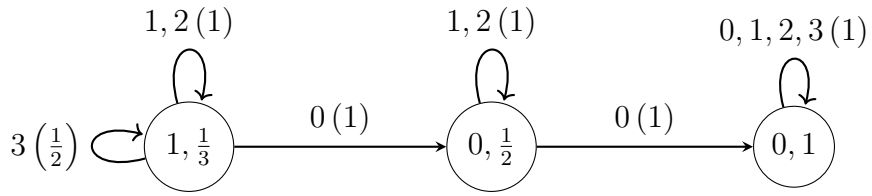
Now we will process the new state q_2. For quadrant $(\psi_2)_0$ we create a new state q_3. Both quadrants $(\psi_2)_1$ and $(\psi_2)_2$ are identical to the image in state q_2 and the quadrant $(\psi_2)_3$ contains the zero image, which means there will be no transition from state q_2 labelled 3.

At last, all quadrants of q_3 are identical to the image ψ_3, so there will be four transitions from q_3 to q_3 with labels $0, 1, 2, 3$ all with weight 1.

The initial and final distributions are:

$$I = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \qquad F = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{2} \\ 1 \end{pmatrix}$$

The resulting automaton is



According to the following theorem, the algorithm produces an ap-WFA with the minimal number of states that precisely defines the input image, assuming such an automaton exists. Although the theorem and its proof can be found in article [1], we felt it was crucial to include it here because it emphasizes several important properties of the encoding algorithm.

Theorem 5. *[1]*

- *Multi-resolution image ψ can be generated by a WFA if and only if the multi-resolution images*

  $$(\psi)_w \text{ for all } w \in \Sigma^*$$

  generate a finite-dimensional vector space. The dimension of the vector space is the same as the smallest number of states in any WFA that represents ψ.

- *If ψ can be represented by a WFA, then the algorithm above produces a WFA with the minimum number of states.*

- *If ψ is average preserving then the algorithm produces an average preserving WFA.*

Note that in practice, the algorithm will get as an input only a finite-resolution image of size $2^k \times 2^k$. Therefore, if we go deeper than depth k in the image's quad-tree, we do not get any new information, i.e. we only copy a node's value to all its children. As a consequence, we do not care about nodes deeper than k.

We say that a sub-image ψ_w for some $w \in \Sigma^*$ has depth $d_w = k - |w|$. When the algorithm searches for a linear combination that expresses the sub-image ψ_w (step 2.a), in practice we only need to consider quad-trees up to depth d_w. Therefore, even in images with greater depth than d_w it is enough to only use quad-tree nodes up to depth d_w. We can do that due to the ap property of ψ. Because the algorithm runs in breath-first manner, we are guaranteed that all images available for the linear combination have depth at least d_w.

3.2.1 Smallest WFA

By using the algorithm 3.2 we were guaranteed that the resulting WFA has a minimal number of states, but on the other hand, it is not necessarily minimal with respect to the number of edges. The resulting WFA has a high number of transitions because the last details of the image often require a significant increase in the number of transitions in the WFA. However, we can slightly modify the algorithm 3.2 so that the resulting WFA will not have a minimal number of states, but the sum of its transitions and states will be minimal. The modified algorithm is described in [3].

During processing the image ψ_i in the algorithm 3.2 we have two options how to process its quadrants:

1. try to express the current sub-image with a linear combination of the images which are available now,

2. or create a new state with the quadrant's image and process its quadrants.

In our algorithm, rather than always choosing the option 1. if such a linear combination exists, we will compare the two alternatives, and choose the one that yields a smaller automaton. The size of the automaton will be expressed as a sum of the number of states and the number of transitions.

However, before deciding which alternative to choose, we must completely process the new state created in 2. to do an accurate comparison between the two options. Therefore, we go from processing the states in breath-first order to depth-first order: the new state created by a quadrant is processed before moving on to the next quadrant. For this reason, it only makes sense to make the new algorithm recursive.

The next modification of the algorithm is that a new state is added only after its quadrants are processed. Edges back to states that have not yet been completely processed are problematic since it is not yet clear what those states' images look like. Therefore, the image of the unprocessed state is not available for the linear combination of its sub-quadrants. This new condition also prevents the creation of any loops in the WFA.

Because we do not permit unprocessed states in the linear combination, we need to have at least one processed state at the start of the algorithm so we can calculate the linear combinations of the sub-quadrants. Therefore the WFA has to be initiated with some fixed images $\psi_1, \ldots, \psi_N$, which will be called the *initial basis*. We will use the same images as in [3], which are generated by the automata representing the linearly independent quadratic polynomials 1, x, y, xy, x^2 and y^2 that we talked about in section 2.1.

Recursive Encoding Algorithm

Input: multi-resolution image ψ
Global variables:

n : number of states in the automaton so far, initialized with N
N : number of images in the initial basis
$\psi_1, \ldots \psi_N$: images in the initial basis
ψ_i : image of state i, $1 \leq i \leq n$

make_wfa(ψ_i, max):

1. If $max < 0$ then
   return(∞)
2. Set $cost \leftarrow 0$
3. For quadrants $a = 0, 1, 2, 3$ do:
   a) If $\exists\, r_1, \ldots r_n \in \mathbb{R}$ such that $(\psi_i)_a = r_1\psi_1 + r_2\psi_2 + \ldots + r_n\psi_n$ then
      $cost_1 \leftarrow$ number of non-zero coefficients r_j
      else
      $cost_1 \leftarrow \infty$
   b) Set $n_0 \leftarrow n$, $n \leftarrow n + 1$, $\psi_n = (\psi_i)_a$
      $cost_2 \leftarrow 1 + make\_wfa(\psi_n, min\{max - cost, cost_1\} - 1)$
   c) If $cost_2 \leq cost_1$:
      - $cost \leftarrow cost + cost_2$

      else

- $cost \leftarrow cost + cost_1$
- remove all transitions from states $n_0 + 1, \ldots, n$
- remove all states $n_0 + 1, \ldots, n$
- set $n \leftarrow n_0$
- add transitions from state i to j with label a and weight r_j, for $r_j \neq 0$

4. If $cost \leq max$ then
   return($cost$)
   else
   return(∞)

The new algorithm now consists of method *make_wfa* that is called recursively, and a global state[1] where the resulting WFA is built. The method returns the cost of processing the four quadrants of supplied image ψ_i. However, if the cost is greater than the value max, an infinite cost is returned instead to indicate that no improvement over the target value max was obtained.

Before the first call of the method *make_wfa*, we increment n by 1 (so that now $n = N + 1$) and then create a new state $N + 1$ for the input image ψ. The first call to *make_wfa* is with the image ψ_{N+1} and $max = \infty$.

The most important step of the algorithm is the third step where the algorithm is trying to express the quadrants of the input image. First, we try to express the quadrant as a linear combination of the processed images. Second, we create a new state for the quadrant and recursively try to approximate its quadrants. The costs of both alternatives are stored in the two variables $cost_1$ and $cost_2$. $cost_1$ is the number of non-zero coefficients r_j of the linear combination, which is equal to the number of transitions potentially added to the WFA if the first alternative is chosen. $cost_2$ is the cost of creating a new state and the cost of the recursive processing of its quadrants. In step 3.c the algorithm chooses the alternative with a lower cost.

The max argument serves as a mechanism for stopping the recursion. It gets propagated by the expression $min\{max - cost, cost_1\} - 1$. The first argument of the min function simply propagate the current max value while subtracting the cost of the current recursion step. The subtracted 1 indicates the cost of creating a new state in the automaton. The second argument is only used when a linear combination is found in step 3.a. The recursion is stopped when the processing of the image ψ_n costs more than using the linear combination.

When the recursive algorithm returns and the alternative with adding the transitions is better, we have to remove all the transitions and states added during the recursive call and add the transitions expressing the linear combination found in step 3.a. When the algorithm decides that the better alternative is the creation of the new state instead, the only thing we need to do is adding the cost of creating and processing this state to the total cost.

The initial and final distributions are calculated the same way as in the encoding algorithm 3.2. However, the initial distribution will look different because the first N states belong to the basis. Therefore, every state will have an initial distribution of 0, except for the state representing the input image.

[1] We know that the word *state* usually refers to a state of an automaton, but here we are referring to a state of the algorithm.

3.2.2 Encoding of Colour Images

We talked in section 2.2 about how we can express a colour image as three greyscale images with different initial distributions for each colour component (red, green and blue). However, it is possible to only build one automaton with only one initial distribution. To encode the colour image, we concatenate all three colour layers (the greyscale images of the colour components) to a single greyscale image, where each colour layer occupies one quadrant (see figure 3.2). The remaining quadrant can be left black. This composite image will be used as an input for the encoding algorithm.

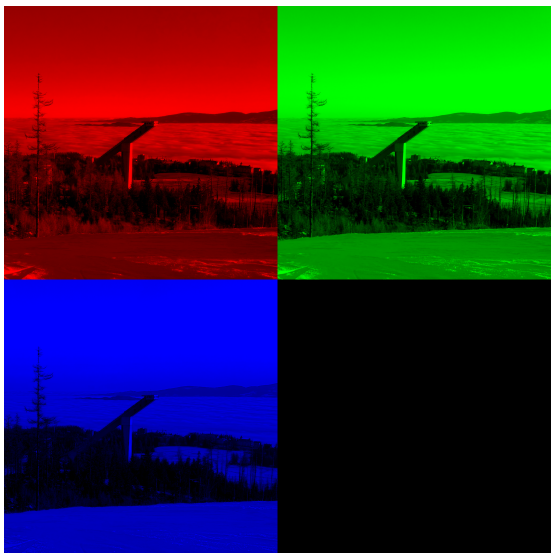


Figure 3.2: Concatenation of colour layers.

This will not change the way we generate images from the resulting WFA, because we can assign each colour component the initial distribution IA_a where $a \in \Sigma$ depends on the quadrant where the colour components' image is located. For example, we see in figure 3.2 that the greyscale image representing the red colour component is located in a quadrant with the address 2. Therefore, when generating, its initial distribution will be $I_R = IA_2$.

4. Implementation

In this chapter, we cover technical problems we encountered when implementing the encoding algorithm (section 3.2.1) and the algorithm for generating images from WFA (section 3.1), and propose solutions for those problems. In the program, we refer to the algorithm of generating images as the **decoding algorithm**.

The program is available in the appendix A.2. The implemented algorithms can be run using the program's command line interface. Installation and input parameters are described in the appendix A.1 of the thesis.

Used Technologies

The application was written in `C#`, because the author of this thesis has the most experience with this programming language. We used the `.Net Core 3.1` framework.

External Dependencies

We built the applicatoin using `Alglib` [9] and `Extreme Optimalization` [10] libraries. Both these libraries are used for numerical computations. They provide a broad set of algorithms for linear algebra that were needed in the application.

4.1 Data Structures

The most important data structure of the application is the data structure representing the weighted finite automata. If we imagine the WFA as an oriented graph where transitions are edges and states are the vertices, the best option is to represent the WFA as an **adjacency list** of the transitions — a list of all transitions with non-zero weight. This representation of WFA provides us with better manipulation of the transitions and states, mainly during the encoding algorithm. Also, the encoding algorithm we used creates WFA with very few non-zero weights, so this representation is more memory-efficient than an adjacency matrix.

Unfortunately, the decoding algorithm needs the matrix representation of WFA. For this purpose, we have to convert the adjacency list to adjacency matrices.

4.2 Decoding Algorithm

As we mentioned in section 2.2, each colour component has its own initial distribution. When decoding a colour image using the decoding algorithm from section 3.1, we need to calculate IA_w for $w \in \Sigma^{\leq \lfloor \frac{k}{2} \rfloor}$ for each colour component separately. On the other hand, $A_v F$ for $v \in \Sigma^{\leq \lceil \frac{k}{2} \rceil}$ is the same for all colour components — they do not depend on the initial distribution.

4.3 Encoding Algorithm

The biggest challenge in implementing the encoding algorithm from section 3.2.1 was to implement the step 3.a, where we try to find a linear combination of the state images.

In the first iterations of the algorithm, we are essentially trying to find a solution for a system of linear equations with many more equations than variables. Such systems rarely have an exact solution. Therefore, we tried to use the least-squares method [11] to get an approximate solution. However, because of approximation errors, the least-squares method did not find exact solutions if they existed. Moreover, the recursion stopped too early and the resulting WFA were very imprecise representations of the input images.

We tried to solve this problem by searching for a single number $r_j \in \mathbb{R}$ such that $(\psi_i)_a = r_j \cdot \psi_j$. If we succeeded, we would use it instead of the least-squares result. Unfortunately, this approach was time-consuming and the program was therefore very slow. We tried restricting this step only to images larger then or as large as the image being processed. Unfortunately, this simplification did not yield any significant speed-up.

After several more attempts to improve this step of the algorithm, we arrived at the following implementation:

- we try to find r_j only for a fixed number of images,
- if we do not succeed, we use the least-squares method, where we use the result only if the error is lower than some threshold,
- otherwise, we say there is no linear combination and proceed deeper into the recursion.

Conclusion

In this thesis, we started by defining an addressing scheme for finite and multi-resolution images. Then we introduced weighted finite automata as a generalization of non-deterministic finite automata. Later, we have shown how WFA can represent multi-resolution images. Finally, we described algorithms for generating images from WFA and for encoding images into WFA and provided an implementation of those algorithms.

We have learned that WFA can have some nice mathematical properties, for example the average preserving property. Also, they allow interesting manipulations with images they represent. Their ability to represent multi-resolution images can have practical applications, especially when we need the same image in many different sizes.

Unfortunately, when implementing the encoding algorithm we found out that the pure version is not computationally feasible. We tried to come up with some optimizations, which allowed us to use the algorithm at least for small images. However, for practical applications, even more optimizations are needed.

For full transparency, we would like to note that the program was also used as a final project for a programming lecture at the Charles University.

Bibliography

[1] Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. *Computers & Graphics*, 17(3):305–313, 1993.

[2] Karel Culik and Jarkko Kari. Digital images and formal languages. In *Handbook of formal languages*, pages 599–616. Springer, 1997.

[3] Jarkko Kari. Image processing using finite automata. *Recent Advances in Formal Languages and Applications*, 25:171–208, 2006.

[4] Karel Culik II, Jarkko Kari, et al. Computational fractal geometry with wfa. *Acta Informatica*, 34(2):151–166, 1997.

[5] Jürgen Albert and Jarkko Kari. Digital image compression. In *Handbook of weighted automata*, pages 453–479. Springer, 2009.

[6] Wikipedia contributors. Rgb color model — Wikipedia, the free encyclopedia, 2022. [Online; accessed 20-July-2022].

[7] Ullrich Hafner. Asymmetric coding in (m)-wfa image compression. *preprint*, 1995.

[8] Karel Culik, II and Juhani Karhumäki. Finite automata computing real functions. *SIAM Journal on Computing*, 23(4):789–814, 1994.

[9] ALGLIB LTD. Alglib, 1999. [Online; accessed 26-Jun-2022].

[10] Extreme Optimization. Extreme optimization numerical libraries for .net, 2003. [Online; accessed 19-July-2022].

[11] Wikipedia contributors. Least squares — Wikipedia, the free encyclopedia, 2022. [Online; accessed 21-July-2022].

List of Figures

A. Attachments

A.1 User Documentation

In this attachment, we will describe how to install the program and use it.

A.1.1 Installation

Run the installation package `Installer.msi` included in the attachment A.2, directory to install WFA_Convertor on your computer. Follow the installer's instructions. Run the WFA_Convertor.exe to launch WFA_Convertor after the installer is finished.

After the application is installed and started the user has to enter the parameters for encoding and decoding. The first is for encoding images into WFA, the letter is for generating images from WFA.

A.1.2 Encoding Image

The encoding algorithm will take an image given by the user and convert it into WFA.

Input Parameters

The parameters for encoding an image must bein the following format:

`encode <image>`.

where

- the keyword `encode` indicates that you have chosen the encoding mode,
- `<image>` is a path to the image file. The path can be either absolute or relative to the application's location in the file system. The program supports images in the following formats: BMP, JPG, PNG, TIFF. If the image is in a different format, the program will either not be able to process it and will write out an error message, or the resulting WFA may not represent the image.

For example, a correct input for encoding is

```
encode C:\Users\MyName\Desktop\pictures\garden.png
```

Because the algorithm builds a WFA that represents the image exactly, the time for encoding can take significant time (even tens of minutes, depending on the image size and complexity). That is why we advise you to encode only images with a size at most 512×512 px.

Output

The resulting automaton will be saved in the same directory as the input image, with the same name but with the suffix `.wfa`. If there already is a WFA file with the same name, the program will ask you, if you want to overwrite the existing file. If not, the program will ask you to enter a new name of the file. Please, just write the name without the directory path or suffix.

A.1.3 Decoding Image

The decoding algorithm generates an image from an input WFA.

Input Parameters

The parameters for decoding an image must bein the following format:

`decode <WFAFile> [depth]`

where

- keyword `decode` indicates, that you have chosen the decoding mode.
- `<WFAFile>`is a path to the WFA file. The path can be either absolute or relative to the application's location in the file system.
- [**`depth`**] is an optional parameter. The parameter indicates how deep into the WFA the algorithm goes. If the parameter is not supplied, the decoder uses the native resolution of the encoded image.

  The parameter must be in the format `d=<value>`. The `<value>` must be a positive integer number determining the chosen depth. If the value has a the wrong format or is negative, the program will write an error message.

For example, a correct input for decoding is

```
decode C:\Users\MyName\Desktop\pictures\garden.wfa d=8
```

Output

The output is a decoded PNG image. It will be saved in the same directory as the input WFA file with the same name as the WFA. If there already is an image with the same name, the program will ask you, if you want to overwrite the existing image. If not, the program will ask you to enter a new name of the image. Please, just write the name without the directory path or suffix.

A.2 Software and Electronic Attachments

This attachment is included in electronic format as an archive. Below is the structure of the archive.

```
root
├── WFA_Convertor
│   ├── Installer - project for building a Windows installer
│   ├── WFA_Compression - C# application WFA_Convertor
│   ├── WFA_Lib - C# library for data structures and algorithms
│   └── WFA_Convertor.sln - Visual Studio solution file
├── WFA_Convertor Installer
│   └── Installer.msi - Installer of the WFA_Convertor application
└── README.md - basic startup instructions
```