Weighted Finite Automata and Digital Images

Sai Sandesh
Harisrrinivas
Saksham Maitri

Indian Institute of Science

April 10, 2025

Outline

- Black and White
- Operations
- Greyscale
- Inference and De-Inference
- Transformations
- 3D Objects
- Applications

Motivation: A Different Lens on Image Processing

Standard Approaches

Image processing often relies on signal processing techniques (filters, transforms like DCT/Wavelets) or learning-based methods (Convolutional Neural Networks - CNNs).

Motivation: A Different Lens on Image Processing

Standard Approaches

Image processing often relies on signal processing techniques (filters, transforms like DCT/Wavelets) or learning-based methods (Convolutional Neural Networks - CNNs).

An Alternative: Formal Language Theory

- Can we leverage the rigorous mathematical framework of Automata Theory?
- **Challenge:** Images are continuous-valued, Automata are typically discrete.
- **Solution:** Weighted Finite Automata (WFA) bridge this gap by incorporating real-valued weights.

Motivation: A Different Lens on Image Processing

Standard Approaches

Image processing often relies on signal processing techniques (filters, transforms like DCT/Wavelets) or learning-based methods (Convolutional Neural Networks - CNNs).

An Alternative: Formal Language Theory

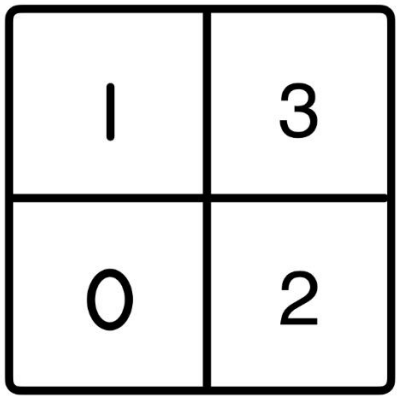
- Can we leverage the rigorous mathematical framework of Automata Theory?
- **Challenge:** Images are continuous-valued, Automata are typically discrete.
- **Solution:** Weighted Finite Automata (WFA) bridge this gap by incorporating real-valued weights.

Our Goal

To understand the fundamentals of WFA image representation and demonstrate its application in transformations, 3D modeling, and feature analysis as presented in the literature.

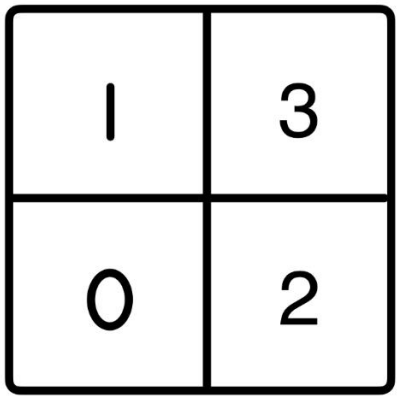
Notations - Address

We have an image with canvas dimension of $2^n \times 2^n$. To address each pixel we define the address as A where A_i=2 $\times$ binary(x_i) + binary(y_i).



Notations - Address

We have an image with canvas dimension of $2^n \times 2^n$. To address each pixel we define the address as A where A_i=2 $\times$ binary(x_i) + binary(y_i).



11	13	31	33
10	12	30	32
01	03	21	23
00	02	20	22

Example: Address Calculation

11	13	31	33
10	12	30	32
01	03	21	23
00	02	20	22

Let us try to find the address of the pixel at $(x, y) = (2, 0)$.

$$x, y \in \{0, 1, 2, \ldots, n-1\}$$

So we take:

$$\text{binary}(x) = \text{bin}(2) = 10, \quad \text{binary}(y) = \text{bin}(0) = 00$$

Now, we compute:

$$2 \times \text{bin}(x) + \text{bin}(y) = 2 \times (10) + (00) = (20) + (00) = 20$$

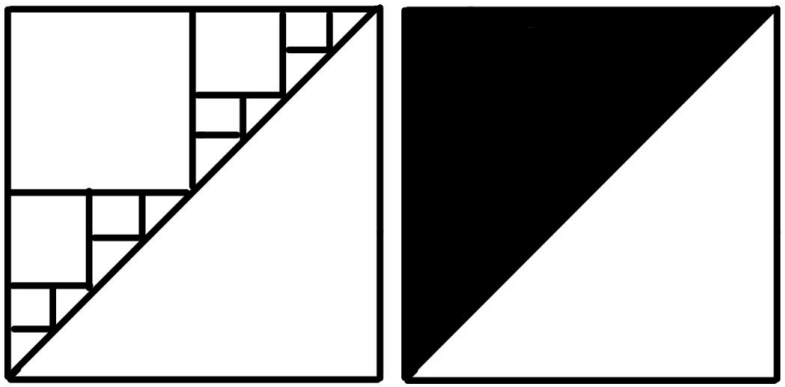
Representing Black and White Images

- We can represent black and white images as FSAs (Finite State Automata). An FSA is similar to NFA (Non-deterministic Finite Automaton).

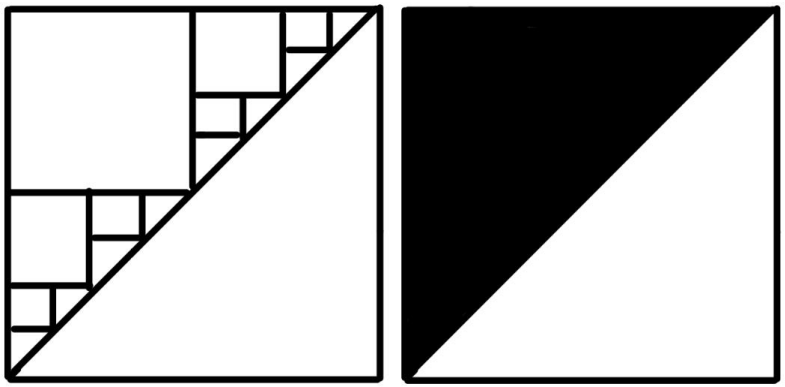
Representing Black and White Images

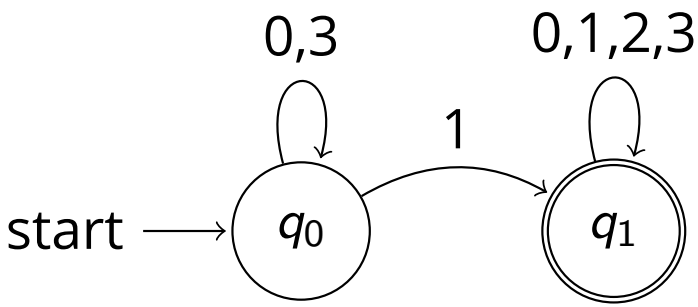
- We can represent black and white images as FSAs (Finite State Automata). An FSA is similar to NFA (Non-deterministic Finite Automaton).
- The language of the FSA is the set of addresses of pixels that are black.

Example 1

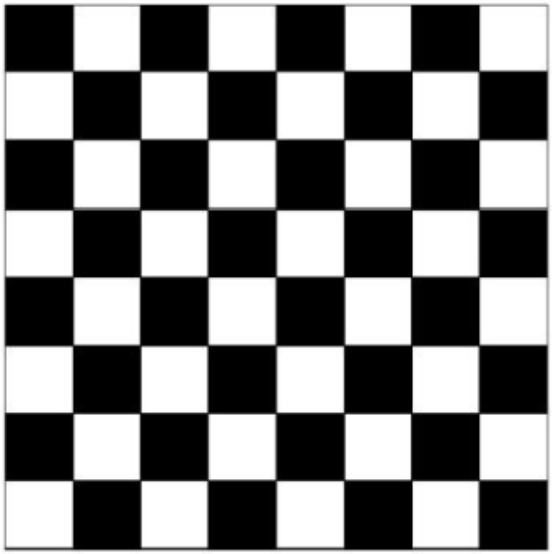


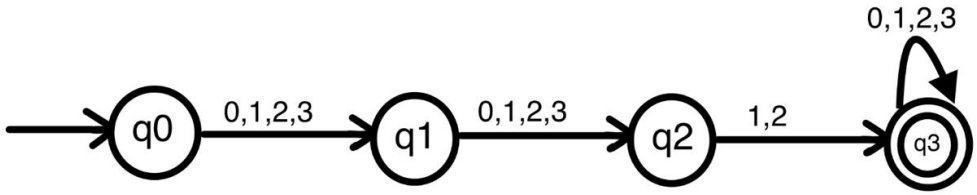
Example 1



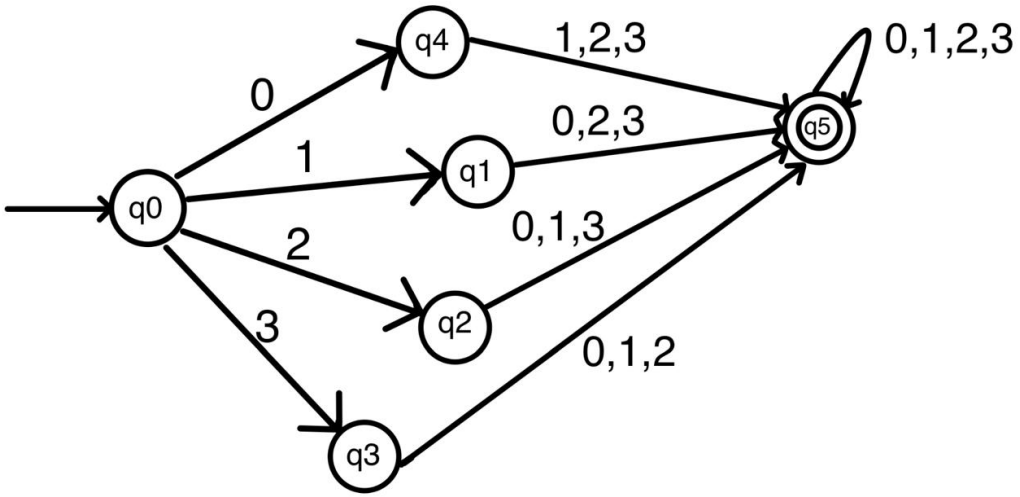


Example 2: Chessboard Automaton

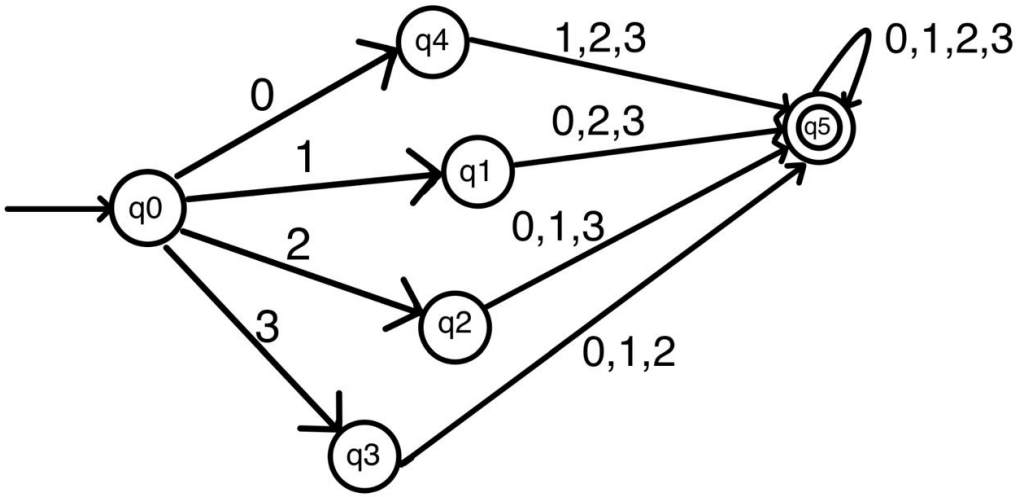


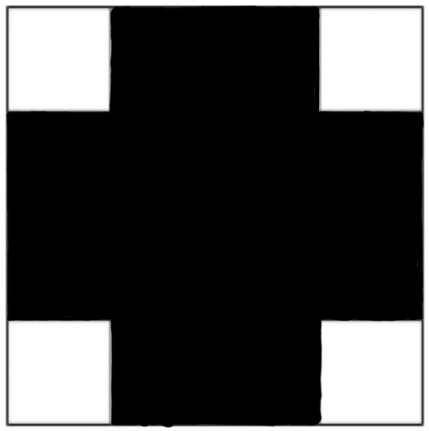


Example 3: Board

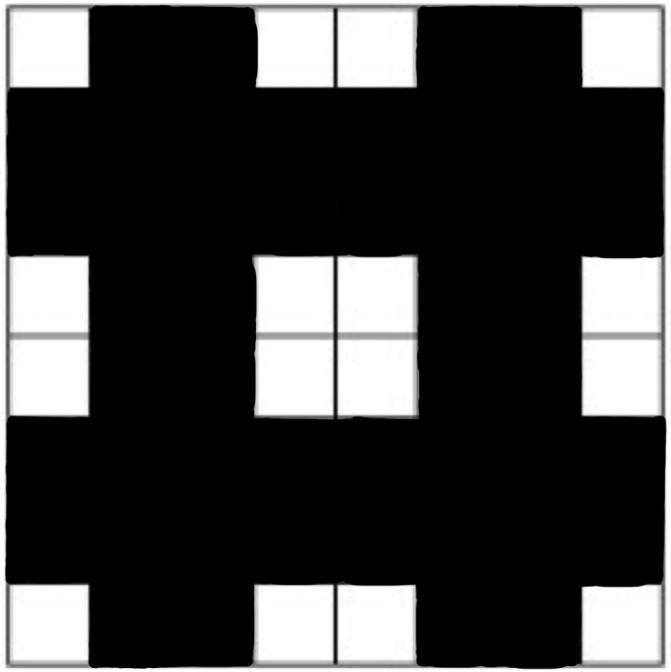


Example 3: Board

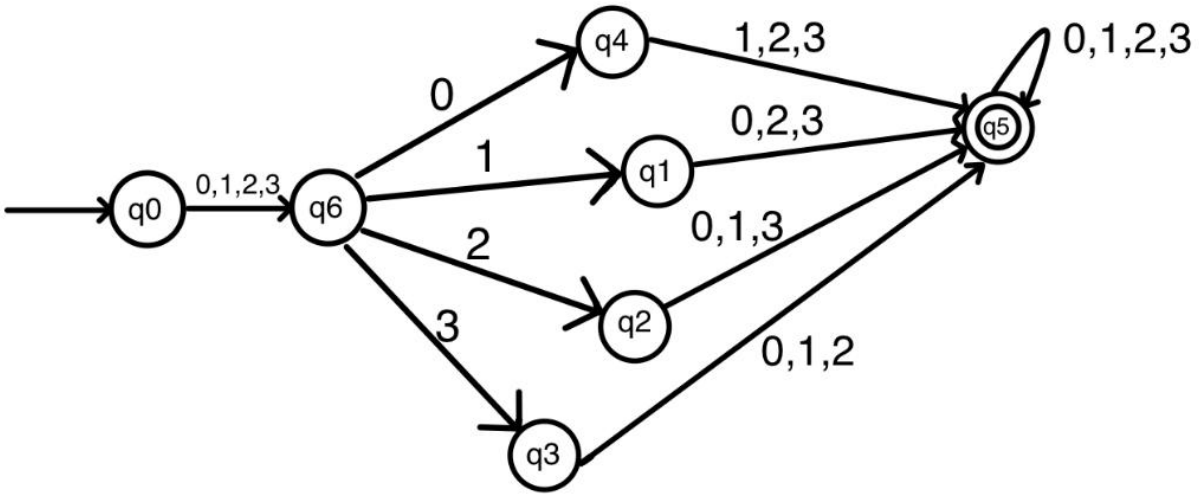




Example 4: Big Board



Example 4: Big Board



Computation

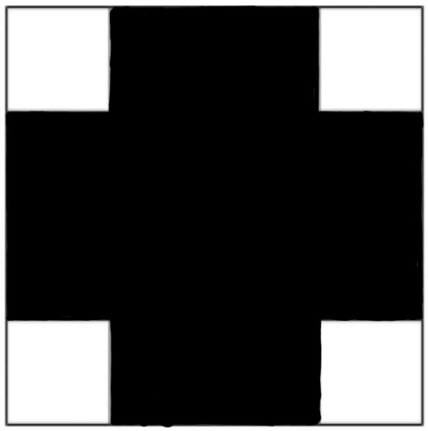
Another way of representing FSA (Finite State Automaton) is by using matrices as follows:

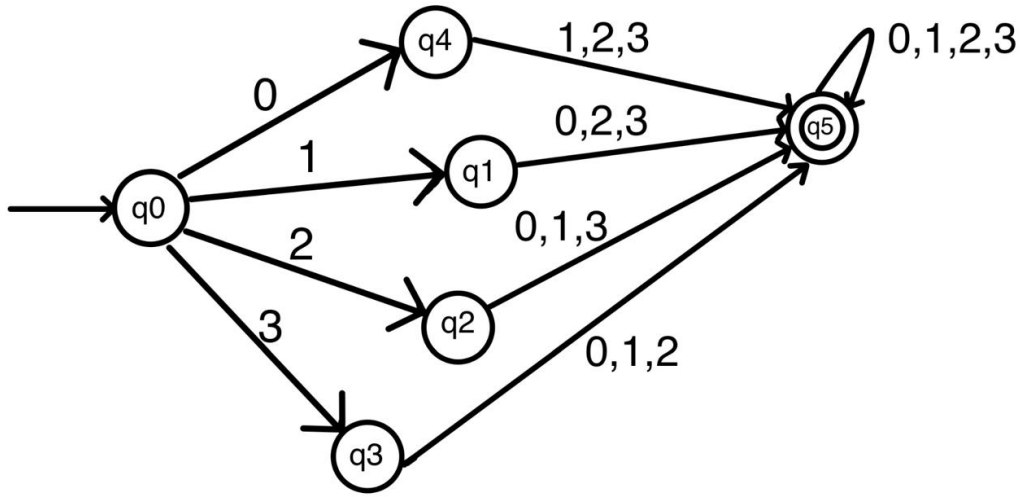
- $\mathbf{I}^A \in \{0,1\}^{1\times m}$ is the **initial distribution**. $\mathbf{I}^A_q = 1$ if q is the initial state, and 0 otherwise.
- $\mathbf{F}^A \in \{0,1\}^{m\times 1}$ is the **final distribution**. $\mathbf{F}^A_q = 1$ if q is a final (accepting) state, and 0 otherwise.
- $\mathbf{W}^A_a \in \{0,1\}^{m\times m}$, for each $a \in \Sigma$, is the **transition matrix**. The entry $\mathbf{W}^A_a[i,j] = 1$ if $j \in \delta(i,a)$, where δ is the transition function. For a string $s = s_1 s_2 s_3 \ldots s_n$, we define:

$$f(s) = \mathbf{I}^A \cdot \mathbf{W}^A_{s_1} \cdot \mathbf{W}^A_{s_2} \cdots \mathbf{W}^A_{s_n} \cdot \mathbf{F}^A$$

This expression evaluates to 1 if the automaton accepts the string s, and 0 otherwise.

Example





Example

We define:

$$\mathbf{I}^A = [1\ 0\ 0\ 0\ 0\ 0] \quad \mathbf{F}^A = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Transition matrices:

$$\mathbf{W}_0^A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{W}_2^A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Computing $f(02)$

We compute:

$$f(02) = \mathbf{I}^A \cdot \mathbf{W}_0^A \cdot \mathbf{W}_2^A \cdot \mathbf{F}^A$$

$$\begin{aligned} \mathbf{I}^A \cdot \mathbf{W}_0^A &= [1\,0\,0\,0\,0\,0] \cdot \mathbf{W}_0^A = [0\,0\,0\,0\,1\,0] \\ [0\,0\,0\,0\,1\,0] \cdot \mathbf{W}_2^A &= [0\,0\,0\,0\,0\,1] \\ [0\,0\,0\,0\,0\,1] \cdot \mathbf{F}^A &= 1 \end{aligned}$$

$$\boxed{f(02) = 1}$$

The FSA accepts the string 02.

What is a Finite State Transducer (FST)?

A Finite State Transducer (FST) represents a transformation from one alphabet Σ_1 to another Σ_2.

Definition

An FST with m states from Σ_1 to Σ_2 is defined by:

- $\mathbf{I} \in \{0,1\}^{1\times m}$: Initial distribution (row vector)
- $\mathbf{F} \in \{0,1\}^{m\times 1}$: Final distribution (column vector)
- $\mathbf{W}_{a,b} \in \{0,1\}^{m\times m}$ for $a \in \Sigma_1$, $b \in \Sigma_2$: Transition matrices

What is a Finite State Transducer (FST)?

A Finite State Transducer (FST) represents a transformation from one alphabet Σ_1 to another Σ_2.

Definition

An FST with m states from Σ_1 to Σ_2 is defined by:

- $\mathbf{I} \in \{0,1\}^{1\times m}$: Initial distribution (row vector)
- $\mathbf{F} \in \{0,1\}^{m\times 1}$: Final distribution (column vector)
- $\mathbf{W}_{a,b} \in \{0,1\}^{m\times m}$ for $a \in \Sigma_1$, $b \in \Sigma_2$: Transition matrices

Application: Transformations like translation(wrapped), scaling, and rotation of black-and-white images.

Transforming an Image: Translation

Given:

- FSA $A = (\mathbf{I}^A, \mathbf{F}^A, \mathbf{W}_a^A)$ for an image
- FST $T = (\mathbf{I}^T, \mathbf{F}^T, \mathbf{W}_{a,b}^T)$

We can obtain a new FSA B representing the transformed image:

$$\mathbf{I}^B = \mathbf{I}^T \otimes \mathbf{I}^A$$

$$\mathbf{F}^B = \mathbf{F}^T \otimes \mathbf{F}^A$$

$$\mathbf{W}_b^B = \sum_{a \in \Sigma} \mathbf{W}_{a,b}^T \otimes \mathbf{W}_a^A \quad \forall b \in \Sigma$$

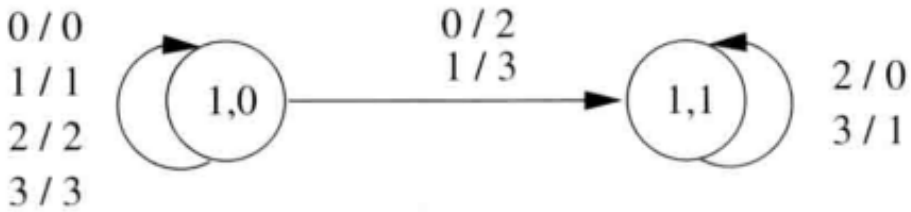
Note: $\otimes$ is the tensor product (Kronecker product) between matrices.

Example FST and Operation

Tensor Product: If $T \in \mathbb{R}^{s \times t}$ and $Q \in \mathbb{R}^{p \times q}$,

$$T \otimes Q = \begin{pmatrix} T_{11}Q & \cdots & T_{1t}Q \\ \vdots & \ddots & \vdots \\ T_{s1}Q & \cdots & T_{st}Q \end{pmatrix}$$

Translation by 1 unit



We are essentially changing the bin(x) from $w01^r$ to $w10^r$ where $0 \leq r \leq n-1$.

Since we made the address using 2*binary(x), we need to change the last 0/1 to 2/3(respectively) such that there is no 0 or 1 after that in the binary expression.

Example

11	13	31	33
10	12	30	32
01	03	21	23
00	02	20	22

Figure 1: 4*4 unit canvas

In the image, lets say we need to translate the address 03 , then according to the FST, it will go to 21.

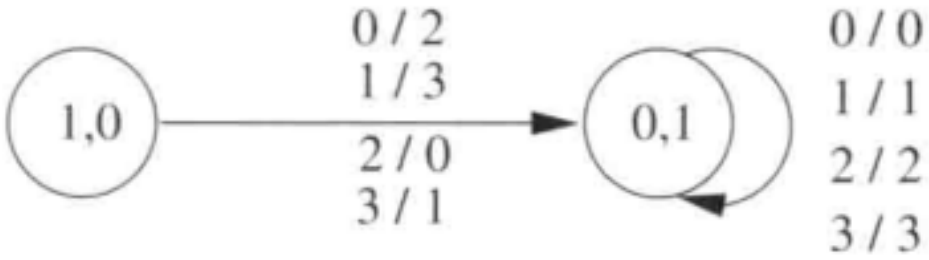
Translation relative to the image

- When we need to translate the image by 1/2 of its size, we mean that we want to translate the image by 2^{n-1} units.
- That means we need to change the most significant digit of binary(x) by 1. If it's 0, we change it to 1 or if it's 1, we change that to 0.
- Same intuition can be applied when we translate the image by 1/4 its size. We need to change the 2 most significant digits of binary(x) as follows:

$$00 \rightarrow 01$$
$$01 \rightarrow 10$$
$$10 \rightarrow 11$$
$$11 \rightarrow 00$$

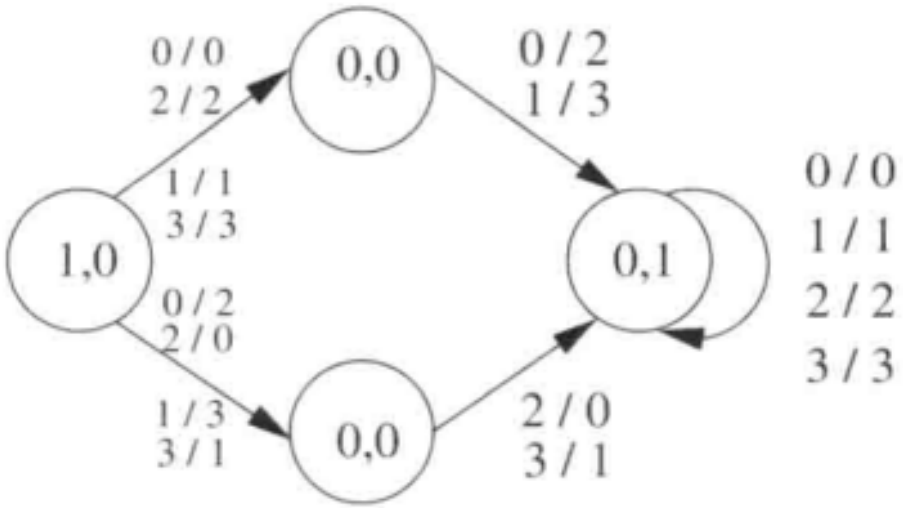
Example image

Translation by 1/2 the image size



FST for Translation by 1/2 Square

Translation by 1/4 the image size



FST for Translation by 1/4 Square

Scaling by 4

11	13	31	33
10	12	30	32
01	03	21	23
00	02	20	22

- Scaling by a factor of k means scaling in both directions by $\sqrt{k}$.
- We dont need any square other than the 4 central one in the scaled image.

New FSA Construction

- **Old FSA**: $(\Sigma, Q, \delta, I, F)$
- **New FSA**: $(\Sigma', Q', \delta', I', F')$

Definitions

- $\Sigma' = \Sigma$
- $Q' = Q \cup \{q_0\}, \quad I' = \{q_0\}, \quad F' = F$
- $\delta'(q_0, 0) = \delta(\delta(q_0, 0), 3)$
- $\delta'(q_0, 1) = \delta(\delta(q_0, 1), 2)$
- $\delta'(q_0, 2) = \delta(\delta(q_0, 2), 1)$
- $\delta'(q_0, 3) = \delta(\delta(q_0, 3), 0)$
- $\delta'(q, a) = \delta(q, a), \quad \forall a \in \Sigma, \quad \forall q \in Q$

Scaling by 1/4

- We can apply similar intuition here to scale the FSA transitions.
- In the FSA, starting from the initial state q_0, apply the following changes:
  - If $\delta(q_0, 0) = p$ then $\delta'(q_0, 03) = p$
  - If $\delta(q_0, 1) = p$ then $\delta'(q_0, 12) = p$
  - If $\delta(q_0, 2) = p$ then $\delta'(q_0, 21) = p$
  - If $\delta(q_0, 3) = p$ then $\delta'(q_0, 30) = p$

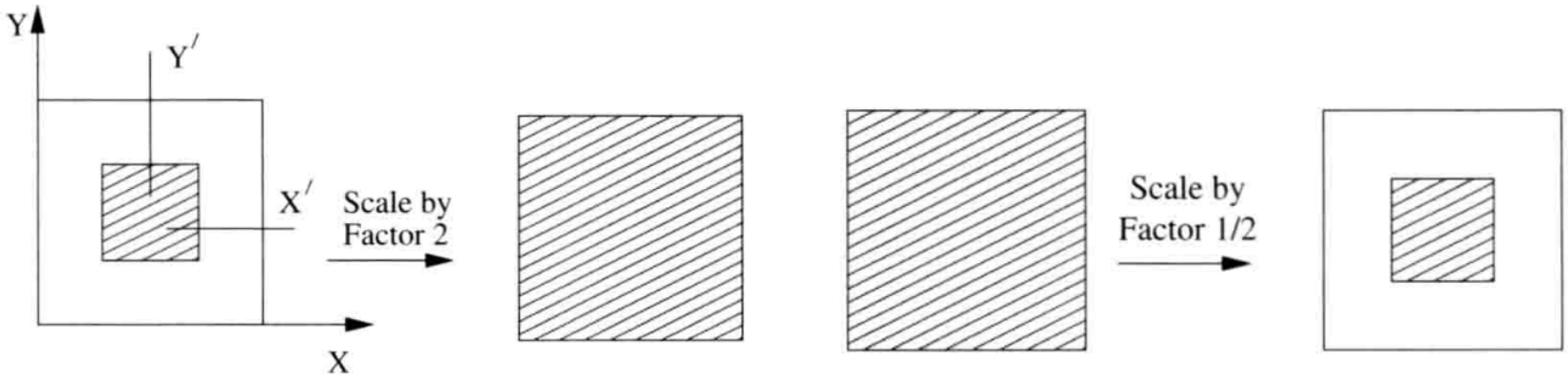
Scaling down by 1/4

- **Old FSA**: $(\Sigma, Q, \delta, s, F)$
- **New FSA**: $(\Sigma', Q', \delta', q', F')$

Definitions

- $\Sigma' = \Sigma$
- $Q' = Q \cup \{q'\} \cup \{q_0, q_1, q_2, q_3\}, \quad I' = \{q'\}, \quad F' = F$
- $\delta'(q', 0) = \{q_0\}, \delta'(q_0, 3) = \{q | \exists i, i \in I, \delta(i, 0) = q\}$
- $\delta'(q', 1) = \{q_1\}, \delta'(q_1, 2) = \{q | \exists i, i \in I, \delta(i, 1) = q\}$
- $\delta'(q', 2) = \{q_2\}, \delta'(q_2, 1) = \{q | \exists i, i \in I, \delta(i, 2) = q\}$
- $\delta'(q', 3) = \{q_3\}, \delta'(q_3, 0) = \{q | \exists i, i \in I, \delta(i, 3) = q\}$
- $\delta'(q, a) = \delta(q, a), \quad \forall a \in \Sigma, \quad \forall q \in Q$

Example



Only Black and White?

But what is the use of a black and white image? We need to represent more than just two colors. Hence, we introduce the concept of **greyscale**, where:

- Each pixel can take on any value between 0 and 1, representing the intensity of the pixel.
- These kind of images can be represented by **weighted finite automata**.
- In reality, we use discrete values between 0 and 255 for the same purpose.

Weighted Finite Automata

Definition

A **weighted finite automaton** (WFA) is a 5-tuple $A = (K, \Sigma, W, I, F)$ where:

- K is a finite set of states.
- Σ is a finite input alphabet.
- W is the set of weight matrices $W_\alpha : K \times K \to \mathbb{R}$ for each $\alpha \in \Sigma \bigcup \{\epsilon\}$.
- $I : K \to \mathbb{R}$ is the initial distribution.
- $F : K \to \mathbb{R}$ is the final distribution.

Analogies with FSAs

- It uses *mostly* the same notation as described in the previous section. e.g., $W_\alpha(p, q)$ is the weight of the transition from state p to state q on input α, given the entry is non-zero.
- Continuing the analogy, the WFA naturally defines a function $f : \Sigma^* \to \mathbb{R}$ given by:

$$f(w) = I \cdot W_{\alpha_1} \cdots W_{\alpha_k} \cdot F$$

where $w = \alpha_1 \cdots \alpha_k$ is a string of length k.

- Note that this function f is not Boolean anymore, but a real-valued function.

Alternate f Definition

There is a natural notion of a **path** in a WFA, which is a 2-tuple ($q_0 q_1 \cdots q_k$, $w = \alpha_1 \cdots \alpha_k$) where $q_0, q_1, \ldots, q_k$ are states in K, and α_i is the input symbol on the transition from q_{i-1} to q_i.

Definition

The **weight** of a path P is defined as:

$$W(P) = I_{q_0} \cdot W_{\alpha_1}(q_0, q_1) \cdots W_{\alpha_k}(q_{k-1}, q_k) \cdot F_{q_k}$$

From the above definition, we obtain an equivalent definition of the function f:

$$f(w) = \sum_{P=(\cdots,w)} W(P)$$

Let us see some examples of WFAs.

One More Important Definition!

Definition

A WFA M is said to be **average preserving** if the function f that it represents, satisfies the following property:

$$f(w) = \frac{1}{|\Sigma|} \sum_{\alpha \in \Sigma} f(w\alpha) \text{ for all } w \in \Sigma^*$$

Lemma

A WFA M is average preserving if and only if the following condition holds:

$$\sum_{\alpha \in \Sigma} W_\alpha \cdot F = |\Sigma| F$$

Representation of Grey-Scale Images

- Grey-scale image: $2^m \times 2^m$ pixels, each with a real grey-ness value.
  In reality, these values are discrete, e.g., 0 to 255.
- Each pixel is addressed by $x \in \Sigma^k$, $\Sigma = \{0, 1, 2, 3\}$, where $1 \leq k \leq m$.
  When $k < m$, x addresses the corresponding subsquare of the image.
- Finite Resolution Image: $f_I : \Sigma^k \to \mathbb{R}$ gives pixel value at address x.
  ($2^k \times 2^k$ pixels)
- Multi-resolution Image: $f_I : \Sigma^* \to \mathbb{R}$.
- A multi-resolution image **must** be average preserving!

$$f_I(x) = \frac{1}{4}\left[f_I(x0) + f_I(x1) + f_I(x2) + f_I(x3)\right].$$

- WFA M represents the image if $f_M = f_I$.
  We will only consider multi-resolution images and their WFAs.

Example: Linear Grey-ness Function

Example

Consider a 2-state WFA with:

- $I = (1, 0)$, $F = \left(\frac{1}{2}, 1\right)$.
- Weight matrices:

$$W_0 = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} \\ 0 & 1 \end{pmatrix}, \quad W_1 = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix}, \quad W_2 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix}, \quad W_3 = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} \\ 0 & 1 \end{pmatrix}.$$

Example: Linear Grey-ness Function

Example

Consider a 2-state WFA with:

- $I = (1, 0)$, $F = \left(\frac{1}{2}, 1\right)$.
- Weight matrices:

$$W_0 = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} \\ 0 & 1 \end{pmatrix}, \quad W_1 = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix}, \quad W_2 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix}, \quad W_3 = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} \\ 0 & 1 \end{pmatrix}.$$

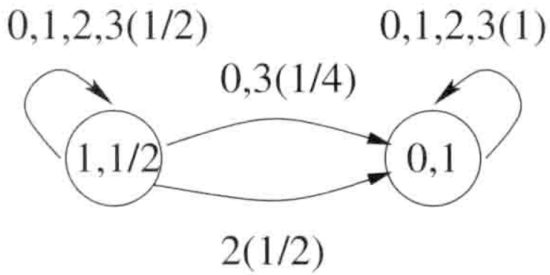


Figure 2: WFA for our Example

Example: Linear Grey-ness Function

Let us compute values of some pixels:

- $f(3) = I \cdot W_3 \cdot F = (1,0) \cdot \begin{pmatrix} \frac{1}{2} & \frac{1}{4} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{2} \\ 1 \end{pmatrix} = \frac{3}{8}.$
- $f(22) = (1,0) \cdot \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{2} \\ 1 \end{pmatrix} = \frac{7}{8}.$

Recall that we can also use the alternate definition of f:

$$\begin{aligned} f(03) &= (1 \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2}) + (1 \times \frac{1}{2} \times \frac{1}{4} \times 1) + (1 \times \frac{1}{4} \times 1 \times 1) \\ &= \frac{1}{8} + \frac{1}{8} + \frac{1}{4} = \frac{1}{2} \end{aligned}$$

Example: Linear Grey-ness Function

- The above WFA, as seen before, represents grey-ness variation across multiple resolutions.

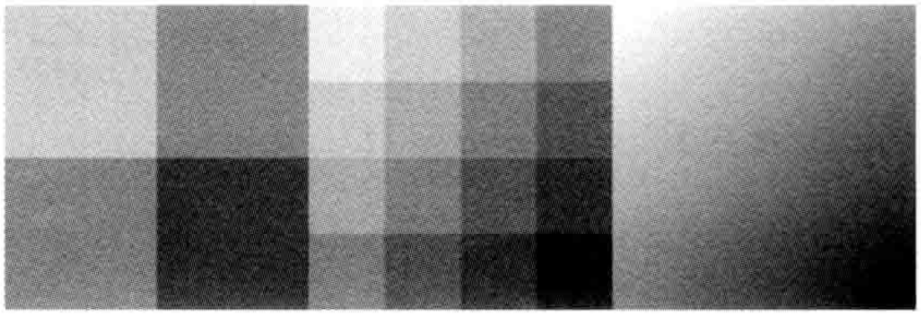


Figure 3: Multi-resolution Images for our Example

Inferencing and De-Inferencing

Now we know that we can represent every digital greyscale image as a WFA.
Now, we provide two algorithms:

- **Inferencing Algorithm:** To inference a WFA, given a (finite resolution) digital image.
- **De-Inferencing Algorithm:** To construct a finite resolution approximation of the multi-resolution image, given by a WFA.

De-Inferencing Algorithm

Algorithm: De_Infer_WFA

Input: WFA $M = (I, F, W_0, W_1, W_2, W_3)$.
Output: $f(x)$ for $x \in \Sigma^k$.

- Step 1: Set $\phi_p(\epsilon) \leftarrow F_p$ for all $p \in Q$.
- Step 2: For $i = 1$ to k:
  - For all $p \in Q$, $x \in \Sigma^{i-1}$, and $\alpha \in \{0, 1, 2, 3\}$,
    Compute: $\phi_p(\alpha x) \leftarrow \sum_{q \in Q} W_\alpha(p, q) \cdot \phi_q(x)$.
- Step 3: For $x \in \Sigma^k$, Compute: $f(x) = \sum_{q \in Q} I_q \cdot \phi_q(x)$.

De-Inferencing Algorithm

Algorithm: De_Infer_WFA

Input: WFA $M = (I, F, W_0, W_1, W_2, W_3)$.
Output: $f(x)$ for $x \in \Sigma^k$.

- Step 1: Set $\phi_p(\epsilon) \leftarrow F_p$ for all $p \in Q$.
- Step 2: For $i = 1$ to k:
  - For all $p \in Q$, $x \in \Sigma^{i-1}$, and $\alpha \in \{0, 1, 2, 3\}$,
    Compute: $\phi_p(\alpha x) \leftarrow \sum_{q \in Q} W_\alpha(p, q) \cdot \phi_q(x)$.
- Step 3: For $x \in \Sigma^k$, Compute: $f(x) = \sum_{q \in Q} I_q \cdot \phi_q(x)$.

Time complexity: $O(n^2 4^k)$, where $n = |Q|$ and $2^k * 2^k$ is the image size.

Inferencing Algorithm

Algorithm: Infer_WFA

Input: Image $\mathcal{I}$ of size $2^k \times 2^k$.
Output: WFA M.

- Step 1: Initialize: $N \leftarrow 0$, $i \leftarrow 0$, $F_{q_0} \leftarrow f_{\text{avg}}(\epsilon)$, $\gamma(q_0) \leftarrow \epsilon$.
- Step 2: For $x = \gamma(q_i)$, and each $\alpha \in \{0, 1, 2, 3\}$:
  - If $f_{x\alpha} = \sum_{j=0}^{N} c_j \phi_j$, set $W_\alpha(q_i, q_j) \leftarrow c_j$.
  - Else, add new state q_{N+1}, set $W_\alpha(q_i, q_{N+1}) \leftarrow 1$, $F_{q_{N+1}} \leftarrow f_{\text{avg}}(x\alpha)$, $\gamma(q_{N+1}) \leftarrow x\alpha$, and $N \leftarrow N + 1$.
- Step 3: Set $i \leftarrow i + 1$ and goto Step 2.
- Step 4: Set $I_{q_0} \leftarrow 1$, $I_{q_j} \leftarrow 0$ for $1 \leq j \leq N$.

Intuition for Notation

- N: Number of states in the output WFA.
- f_x: Sub-image of $\mathcal{I}$ at position x.
- $f_{avg}(x)$: Average pixel value of sub-image f_x.

Intuition for Notation

- N: Number of states in the output WFA.
- f_x: Sub-image of $\mathcal{I}$ at position x.
- $f_{avg}(x)$: Average pixel value of sub-image f_x.
  I'll be using the previous two f's interchangeably for the next example.

Intuition for Notation

- N: Number of states in the output WFA.
- f_x: Sub-image of $\mathcal{I}$ at position x.
- $f_{avg}(x)$: Average pixel value of sub-image f_x.
  I'll be using the previous two f's interchangeably for the next example.
- $\gamma(q)$: Sub-image introduced by state q.
- F_q: Average pixel value of sub-image $\gamma(q)$.

Intuition for Notation

- N: Number of states in the output WFA.
- f_x: Sub-image of $\mathcal{I}$ at position x.
- $f_{avg}(x)$: Average pixel value of sub-image f_x.
  I'll be using the previous two f's interchangeably for the next example.
- $\gamma(q)$: Sub-image introduced by state q.
- F_q: Average pixel value of sub-image $\gamma(q)$.
  I'll be using the previous two... oh nvm.

Example: Inference of WFA

Example

For the linear sloping AP-function f from the earlier Example:

Example: Inference of WFA

Example

For the linear sloping AP-function f from the earlier Example:
q_0: $\gamma(q_0) = \epsilon$, so $F_{q_0} = \frac{1}{2}$, the average pixel value of the entire image.

Example: Inference of WFA

Example

For the linear sloping AP-function f from the earlier Example:
q_0: $\gamma(q_0) = \epsilon$, so $F_{q_0} = \frac{1}{2}$, the average pixel value of the entire image.

- Sub-square 1: This is just $\frac{1}{2} f_\epsilon$, so $W_1(q_0, q_0) = \frac{1}{2}$.

Example: Inference of WFA

Example

For the linear sloping AP-function f from the earlier Example:
q_0: $\gamma(q_0) = \epsilon$, so $F_{q_0} = \frac{1}{2}$, the average pixel value of the entire image.

- Sub-square 1: This is just $\frac{1}{2} f_\epsilon$, so $W_1(q_0, q_0) = \frac{1}{2}$.
- Sub-square 0: New state q_1, $W_0(q_0, q_1) = 1$, $F_{q_1} = \frac{1}{2} = f_{avg}(1)$.

Example: Inference of WFA

Example

For the linear sloping AP-function f from the earlier Example:
q_0: $\gamma(q_0) = \epsilon$, so $F_{q_0} = \frac{1}{2}$, the average pixel value of the entire image.

- Sub-square 1: This is just $\frac{1}{2} f_\epsilon$, so $W_1(q_0, q_0) = \frac{1}{2}$.
- Sub-square 0: New state q_1, $W_0(q_0, q_1) = 1$, $F_{q_1} = \frac{1}{2} = f_{avg}(1)$.
- Sub-square 3: $W_3(q_0, q_1) = 1$.

Example: Inference of WFA

Example

For the linear sloping AP-function f from the earlier Example:
q_0: $\gamma(q_0) = \epsilon$, so $F_{q_0} = \frac{1}{2}$, the average pixel value of the entire image.

- Sub-square 1: This is just $\frac{1}{2} f_\epsilon$, so $W_1(q_0, q_0) = \frac{1}{2}$.
- Sub-square 0: New state q_1, $W_0(q_0, q_1) = 1$, $F_{q_1} = \frac{1}{2} = f_{avg}(1)$.
- Sub-square 3: $W_3(q_0, q_1) = 1$.
- Sub-square 2: $f_2 = 2f_1 - \frac{1}{2} f_\epsilon$, so $W_2(q_0, q_0) = -\frac{1}{2}$, $W_2(q_0, q_1) = 2$.

Example: Inference of WFA

Example

Now, from the previous slide, consider the sub-squares 00, 01, 02, and 03. They can be expressed as:

- $f_{00} = \frac{3}{2}f_1 - \frac{1}{2}f_\epsilon$, $f_{01} = f_1 - \frac{1}{4}f_\epsilon$, $f_{02} = 2f_1 - \frac{3}{4}f_\epsilon$, $f_{03} = \frac{3}{2}f_1 - \frac{1}{2}f_\epsilon$

And so on and so forth.

Example: Inference of WFA

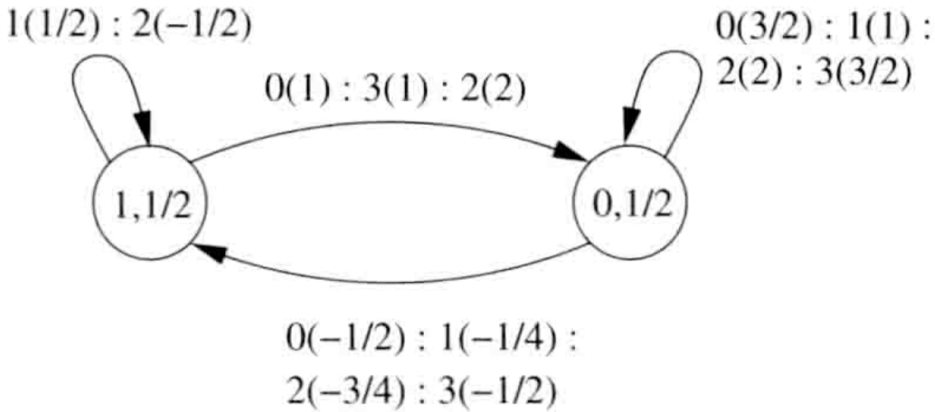


Figure 4: Inferenced WFA for our Example

- The above results in a minimal average-preserving WFA.
- However, we will not prove this fact. Part of the proof uses the fact that states in the WFA are linearly independent.

Approximations and Compression

- Approximate representation: Introduce error δ in the algorithm to reduce states. This gives a WFA with least mean square error (LMS) approximation.
- Grey-scale image: 8 bits/pixel, e.g., 512×512 needs 256 KB.
- WFA with n states: $8n^2$ bytes (in general, we obtain sparse matrices).
- Compression condition: This can be obtained for, say 50% compression for $2^k \times 2^k$ image, by:

$$8n^2 \leq 4^k/2$$
$$n \leq 2^{k-2}$$

- Hence, for a 256x256 image, $n \leq 64$ to obtain any good compression.

WFA Image Representation

- **Recap:** Gray-scale images can be represented by Weighted Finite Automata (WFA).
- A WFA $M = (K, \Sigma, W, I, F)$ computes a function $f : \Sigma^* \to \mathbb{R}$.
- For an image pixel/subsquare with address $x = a_1 a_2 \dots a_k$:

$$f(x) = I \cdot W_{a_1} \cdot W_{a_2} \cdot \dots \cdot W_{a_k} \cdot F$$

- This function $f(x)$ gives the grayness value.
- **Goal:** Perform transformations (scaling, translation, rotation) directly on this WFA representation.
- **Tool:** Weighted Finite Transducers (WFT).

Weighted Finite Transducers (WFT)

Purpose

To define *weighted* relationships between input strings (e.g., original addresses) and output strings (e.g., transformed addresses), potentially modifying values.

Weighted Finite Transducers (WFT)

Purpose

To define *weighted* relationships between input strings (e.g., original addresses) and output strings (e.g., transformed addresses), potentially modifying values.

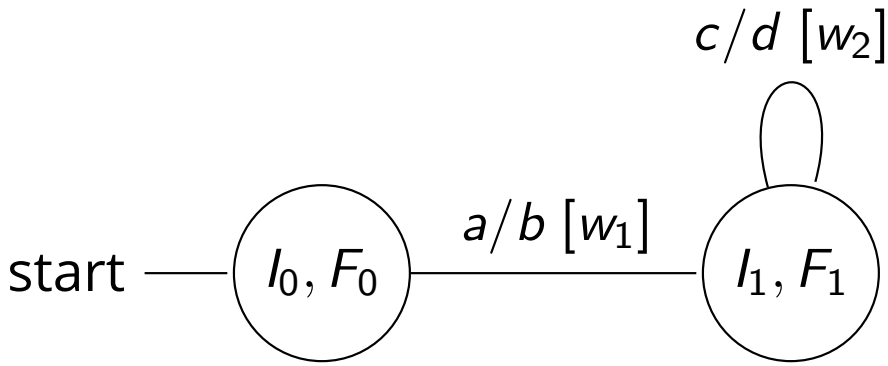
Definition (WFT)

An n-state WFT M over alphabet Σ is specified by:

- Alphabet: Σ (e.g., $\{0, 1, 2, 3\}$)
- Weight Matrices: $W_{a,b} \in \mathbb{R}^{n \times n}$ for each $a, b \in \Sigma \cup \{\epsilon\}$. These matrices hold the weights for transitioning on input a and producing output b.
- Initial Distribution: $I \in \mathbb{R}^{1 \times n}$ (row vector)
- Final Distribution: $F \in \mathbb{R}^{n \times 1}$ (column vector)

Note: ϵ-free WFTs have $W_{\epsilon,\epsilon}$, $W_{a,\epsilon}$, $W_{\epsilon,b}$ as zero matrices for all $a, b \in \Sigma$

Conceptual WFA Diagram



Key Features Illustrated

- States (q_0, q_1) with Initial (I_i) , Final (F_i) distributions.
- Transitions showing Input Symbol / Output Symbol.
- **Crucially:** A numerical Weight ($[w_i]$) associated with each transition.

WFT Function

Definition

A WFT M defines a function $f_M : \Sigma^* \times \Sigma^* \to \mathbb{R}$, called the **weighted relation** between Σ^* and Σ^*, given by:

$$f_M(u, v) = I \cdot W_{u,v} \cdot F, \quad \text{for all } u \in \Sigma^*, v \in \Sigma^*$$

where I is the initial distribution, F is the final distribution, and $W_{u,v}$ is the weight matrix for the pair (u, v).

WFT Function

Definition

A WFT M defines a function $f_M : \Sigma^* \times \Sigma^* \to \mathbb{R}$, called the **weighted relation** between Σ^* and Σ^*, given by:

$$f_M(u, v) = I \cdot W_{u,v} \cdot F, \quad \text{for all } u \in \Sigma^*, v \in \Sigma^*$$

where I is the initial distribution, F is the final distribution, and $W_{u,v}$ is the weight matrix for the pair (u, v).

Definition of $W_{u,v}$

The matrix $W_{u,v}$ is defined as the sum over all possible decompositions:

$$W_{u,v} = \sum_{\substack{a_1 \dots a_k = u \\ b_1 \dots b_k = v}} W_{a_1,b_1} \cdot W_{a_2,b_2} \cdot \dots \cdot W_{a_k,b_k} \qquad (1)$$

Explanation:

Convergence

The function $f_M(u, v)$ is only defined if the sum in the definition of $W_{u,v}$ (1) **converges**. If the sum does not converge, $f_M(u, v)$ remains undefined.

- **Summation Scope:** The sum in (1) is taken over *all possible* ways to decompose the input string u and the output string v simultaneously into sequences of symbols $a_1 \dots a_k$ and $b_1 \dots b_k$, where each $a_i, b_i \in \Sigma \cup \{\epsilon\}$.

Special Case: ϵ-free Transducers

If the WFT M is ϵ-free (no epsilon transitions involved):

- The formula simplifies for strings of equal length k:

$$f_M(a_1 \dots a_k, b_1 \dots b_k) = I \cdot W_{a_1,b_1} \cdot W_{a_2,b_2} \cdot \dots \cdot W_{a_k,b_k} \cdot F$$

- The function value is zero if the input and output strings have different lengths:

$$f_M(u, v) = 0, \quad \text{if } |u| \neq |v|$$

Recall:

WFA Multi-Resolution Function

Remember that a Weighted Finite Automaton (WFA) defines a multi-resolution function $f : \Sigma^* \to \mathbb{R}$.
For an input string $x = \alpha_1\alpha_2 \ldots \alpha_k$, the function value is computed by:

$$f(x) = \sum_{\text{paths for } x} (I \cdot W_{\alpha_1} \cdot W_{\alpha_2} \cdot \cdots \cdot W_{\alpha_k} \cdot F)$$

- The summation is over all possible accepting paths labeled by x in the WFA.
- If the WFA is deterministic, there's only one path and the sum is not needed.
- The operation '$\cdot$' denotes standard matrix multiplication.
- This function $f(x)$ usually represents the grayness or color value at the pixel/subsquare addressed by x.

Applying WFT to WFA Image Function

The Core Operation: Transforming the Image Function

We want to apply a transformation, represented by a weighted relation ρ (or specifically, the relation f_M from a WFT M), to an image function f (represented by a WFA Γ). The goal is to compute the new, transformed image function g.

Applying WFT to WFA Image Function

The Core Operation: Transforming the Image Function

We want to apply a transformation, represented by a weighted relation ρ (or specifically, the relation f_M from a WFT M), to an image function f (represented by a WFA Γ). The goal is to compute the new, transformed image function g.

Definition (Application of ρ to f)

Let $\rho : \Sigma^* \times \Sigma^* \to \mathbb{R}$ be a weighted relation and $f : \Sigma^* \to \mathbb{R}$ be a multi-resolution function (from a WFA). The application of ρ to f, denoted $g = \rho(f)$, is the multi-resolution function $g : \Sigma^* \to \mathbb{R}$ defined by:

$$g(v) = \sum_{u \in \Sigma^*} f(u)\rho(u, v), \quad \text{for all } v \in \Sigma^* \qquad (2)$$

provided the sum converges.

Constructing WFA $\Gamma' = M(\Gamma)$

The Construction (for ϵ-free WFT)

Given:

- An ϵ-free n-state WFT $M = (K_M, \Sigma, W^M, I_M, F_M)$
- An m-state WFA $\Gamma = (K_\Gamma, \Sigma, W^\Gamma, I_\Gamma, F_\Gamma)$ representing function f.

We construct the mn-state WFA $\Gamma' = (K', \Sigma, W', I', F')$ representing $g = M(f)$ as follows:

Constructing WFA $\Gamma' = M(\Gamma)$

The Construction (for ϵ-free WFT)

Given:

- An ϵ-free n-state WFT $M = (K_M, \Sigma, W^M, I_M, F_M)$
- An m-state WFA $\Gamma = (K_\Gamma, \Sigma, W^\Gamma, I_\Gamma, F_\Gamma)$ representing function f.

We construct the mn-state WFA $\Gamma' = (K', \Sigma, W', I', F')$ representing $g = M(f)$ as follows:

- **States K':** $K_M \times K_\Gamma$ (Cartesian product of states)

Constructing WFA $\Gamma' = M(\Gamma)$

The Construction (for ϵ-free WFT)

Given:

- An ϵ-free n-state WFT $M = (K_M, \Sigma, W^M, I_M, F_M)$
- An m-state WFA $\Gamma = (K_\Gamma, \Sigma, W^\Gamma, I_\Gamma, F_\Gamma)$ representing function f.

We construct the mn-state WFA $\Gamma' = (K', \Sigma, W', I', F')$ representing $g = M(f)$ as follows:

- **States K':** $K_M \times K_\Gamma$ (Cartesian product of states)
- **Initial Distribution I':** $I_M \otimes I_\Gamma$

Constructing WFA $\Gamma' = M(\Gamma)$

The Construction (for ϵ-free WFT)

Given:

- An ϵ-free n-state WFT $M = (K_M, \Sigma, W^M, I_M, F_M)$
- An m-state WFA $\Gamma = (K_\Gamma, \Sigma, W^\Gamma, I_\Gamma, F_\Gamma)$ representing function f.

We construct the mn-state WFA $\Gamma' = (K', \Sigma, W', I', F')$ representing $g = M(f)$ as follows:

- **States K':** $K_M \times K_\Gamma$ (Cartesian product of states)
- **Initial Distribution I':** $I_M \otimes I_\Gamma$
- **Final Distribution F':** $F_M \otimes F_\Gamma$

Constructing WFA $\Gamma' = M(\Gamma)$

The Construction (for ϵ-free WFT)

Given:

- An ϵ-free n-state WFT $M = (K_M, \Sigma, W^M, I_M, F_M)$
- An m-state WFA $\Gamma = (K_\Gamma, \Sigma, W^\Gamma, I_\Gamma, F_\Gamma)$ representing function f.

We construct the mn-state WFA $\Gamma' = (K', \Sigma, W', I', F')$ representing $g = M(f)$ as follows:

- **States K':** $K_M \times K_\Gamma$ (Cartesian product of states)
- **Initial Distribution I':** $I_M \otimes I_\Gamma$
- **Final Distribution F':** $F_M \otimes F_\Gamma$
- **Weight Matrices W'_b (for each output symbol $b \in \Sigma$):**

$$W'_b = \sum_{a \in \Sigma} (W^M_{a,b} \otimes W^\Gamma_a)$$

Result of the Automation Construction

Key Outcome

The new WFA Γ', constructed using the tensor product operations described on the previous slide, precisely computes the transformed image function g.

- The function computed by Γ', let's call it $f_{\Gamma'}$, is identical to the function $g = M(f)$ defined by the pixel-level summation (2):

$$f_{\Gamma'}(v) = g(v) = \sum_{u \in \Sigma^*} f(u) f_M(u, v)$$

- This construction allows performing transformations efficiently in the "compressed" WFA domain without explicitly calculating pixel values.

Result of the Automation Construction

Key Outcome

The new WFA Γ', constructed using the tensor product operations described on the previous slide, precisely computes the transformed image function g.

- The function computed by Γ', let's call it $f_{\Gamma'}$, is identical to the function $g = M(f)$ defined by the pixel-level summation (2):

$$f_{\Gamma'}(v) = g(v) = \sum_{u \in \Sigma^*} f(u) f_M(u, v)$$

- This construction allows performing transformations efficiently in the "compressed" WFA domain without explicitly calculating pixel values.

Next: Linearity

This transformation process using WFTs has useful mathematical properties, such as linearity.

WFT Properties

Linearity

WFTs act as **linear operators** on the space of functions representable by WFAs:

$$M(r_1 f_1 + r_2 f_2) = r_1 M(f_1) + r_2 M(f_2)$$

where f_1, f_2 are functions represented by WFAs Γ_1, Γ_2.

WFT Properties

Linearity

WFTs act as **linear operators** on the space of functions representable by WFAs:

$$M(r_1 f_1 + r_2 f_2) = r_1 M(f_1) + r_2 M(f_2)$$

where f_1, f_2 are functions represented by WFAs Γ_1, Γ_2.

Examples using WFTs

Many standard image transformations can be implemented:

- **Scaling:** Modify addresses and potentially values.
- **Translation:** Shift pixels, often using WFTs analogous to FSTs used for B&W images.
- **Rotation:** More complex, often combined with scaling.

Scaling Example: Colour Image

Concept

Scaling transformations modify the size of the image represented by the WFA(s). Similar to 2D BW or 3D FSA scaling, this involves constructing a new automaton where transitions correspond to scaled addresses/paths. For colour images, the scaling construction is applied independently to each R, G, B channel WFA.

Scaling Example: Colour Image

Concept

Scaling transformations modify the size of the image represented by the WFA(s). Similar to 2D BW or 3D FSA scaling, this involves constructing a new automaton where transitions correspond to scaled addresses/paths. For colour images, the scaling construction is applied independently to each R, G, B channel WFA.

Original Image

Scaling Example: Colour Image

Concept

Scaling transformations modify the size of the image represented by the WFA(s). Similar to 2D BW or 3D FSA scaling, this involves constructing a new automaton where transitions correspond to scaled addresses/paths. For colour images, the scaling construction is applied independently to each R, G, B channel WFA.

Original Image

Scale factor of 4

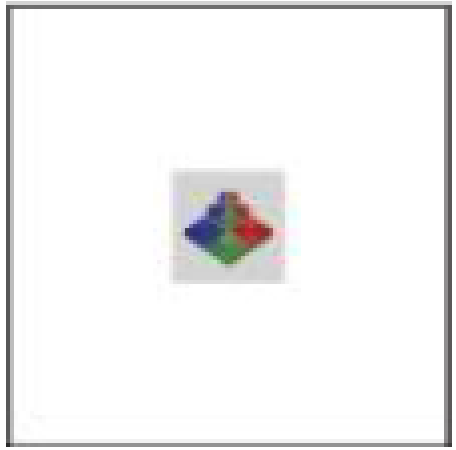
Scaling Example: Colour Image

Concept

Scaling transformations modify the size of the image represented by the WFA(s). Similar to 2D BW or 3D FSA scaling, this involves constructing a new automaton where transitions correspond to scaled addresses/paths. For colour images, the scaling construction is applied independently to each R, G, B channel WFA.

Original Image

Scale factor of 4



- Scaling constructions are applied to each colour channel WFA ($\Gamma_R, \Gamma_G, \Gamma_B$).

Translation Examples: Fractional Shifts (Gray-Scale)

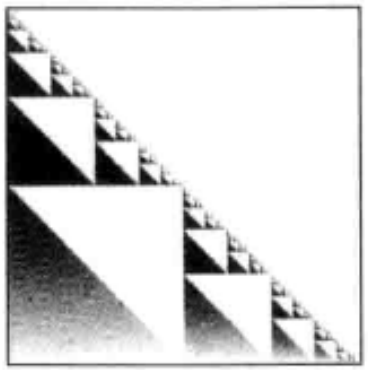
Concept

Translation by large fractions (e.g., 1/2 or 1/4 of the image width/height) is achieved using FSTs/WFTs that modify the Most Significant Bits (MSBs) of the pixel addresses. This effectively shifts the image across major subdivisions of the space.

Translation Examples: Fractional Shifts (Gray-Scale)

Concept

Translation by large fractions (e.g., 1/2 or 1/4 of the image width/height) is achieved using FSTs/WFTs that modify the Most Significant Bits (MSBs) of the pixel addresses. This effectively shifts the image across major subdivisions of the space.

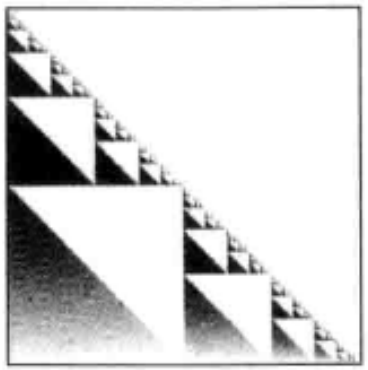


Original Image

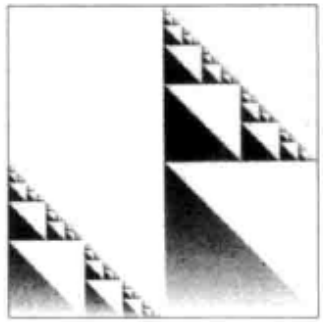
Translation Examples: Fractional Shifts (Gray-Scale)

Concept

Translation by large fractions (e.g., 1/2 or 1/4 of the image width/height) is achieved using FSTs/WFTs that modify the Most Significant Bits (MSBs) of the pixel addresses. This effectively shifts the image across major subdivisions of the space.



Original Image

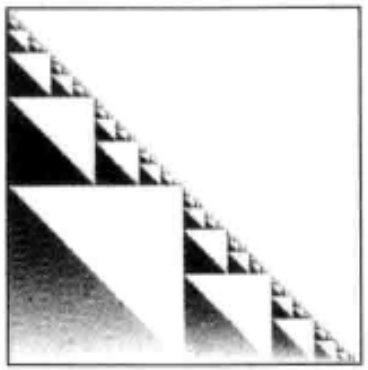


Translated by
1/2 of the square

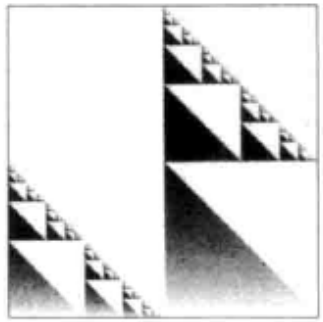
Translation Examples: Fractional Shifts (Gray-Scale)

Concept

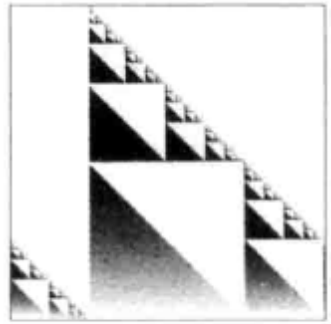
Translation by large fractions (e.g., 1/2 or 1/4 of the image width/height) is achieved using FSTs/WFTs that modify the Most Significant Bits (MSBs) of the pixel addresses. This effectively shifts the image across major subdivisions of the space.



Original Image



Translated by
1/2 of the square

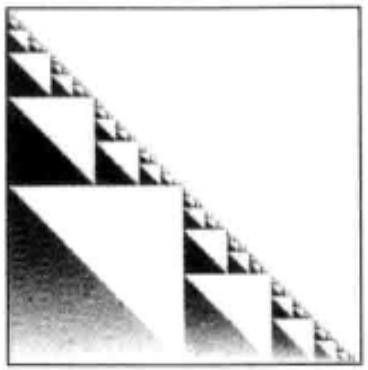


Translated by
1/4 of the square

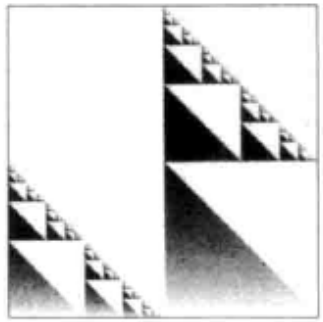
Translation Examples: Fractional Shifts (Gray-Scale)

Concept

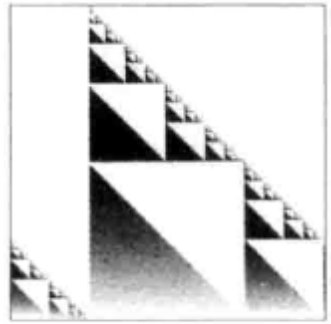
Translation by large fractions (e.g., 1/2 or 1/4 of the image width/height) is achieved using FSTs/WFTs that modify the Most Significant Bits (MSBs) of the pixel addresses. This effectively shifts the image across major subdivisions of the space.



Original Image



Translated by
1/2 of the square



Translated by
1/4 of the square

- Translation by 1/2 flips the single MSB of the relevant coordinate(s).
- Translation by 1/4 modifies the top two MSBs.

Translation of Colour Images

Representing Colour Images

- Colour images (e.g., in RGB format) store separate values for Red, Green, and Blue channels per pixel.
- Using WFAs: Represent a colour image with **three separate WFAs**:
  - Γ_R for the Red channel intensities.
  - Γ_G for the Green channel intensities.
  - Γ_B for the Blue channel intensities.

Translation of Colour Images

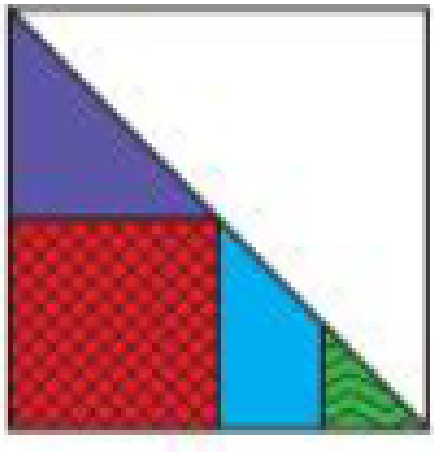
Representing Colour Images

- Colour images (e.g., in RGB format) store separate values for Red, Green, and Blue channels per pixel.
- Using WFAs: Represent a colour image with **three separate WFAs**:
  - Γ_R for the Red channel intensities.
  - Γ_G for the Green channel intensities.
  - Γ_B for the Blue channel intensities.

Applying Transformations (e.g., Translation)

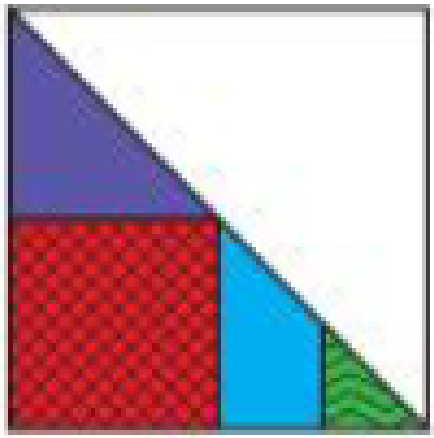
- The **same** FST or WFT (M_{trans}) used for gray-scale translation is applied **independently** to each colour channel's WFA:
  - $\Gamma'_R = M_{trans}(\Gamma_R)$
  - $\Gamma'_G = M_{trans}(\Gamma_G)$
  - $\Gamma'_B = M_{trans}(\Gamma_B)$
- The resulting three WFAs ($\Gamma'_R, \Gamma'_G, \Gamma'_B$) together represent the translated colour image.

Example: Translation of a Colour Image

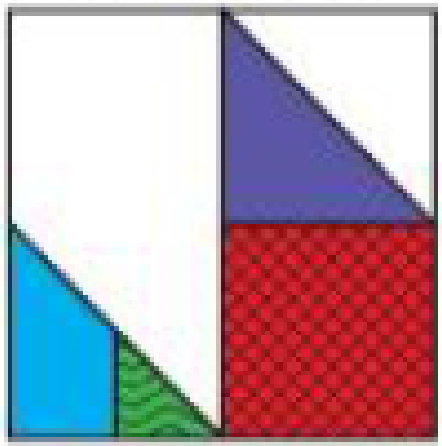


Original Image

Example: Translation of a Colour Image

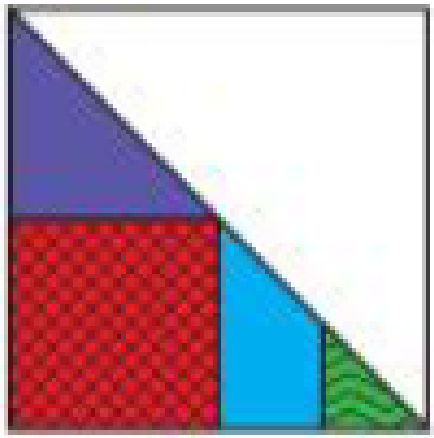


Original Image

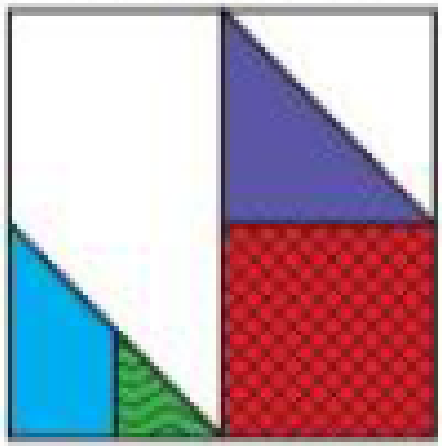


Translated by 1/2 Square

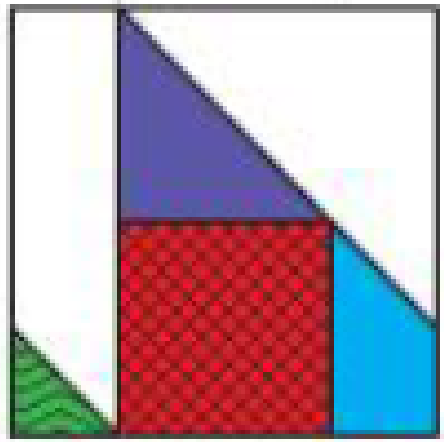
Example: Translation of a Colour Image



Original Image

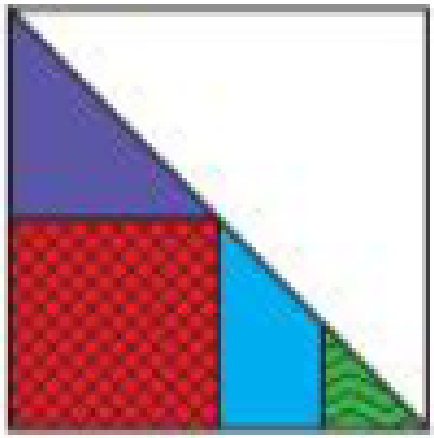


Translated by 1/2 Square

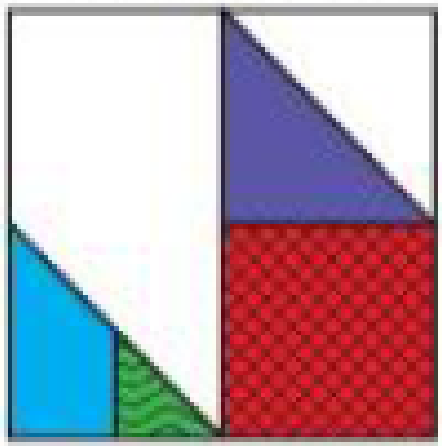


Translated by 1/4 Square

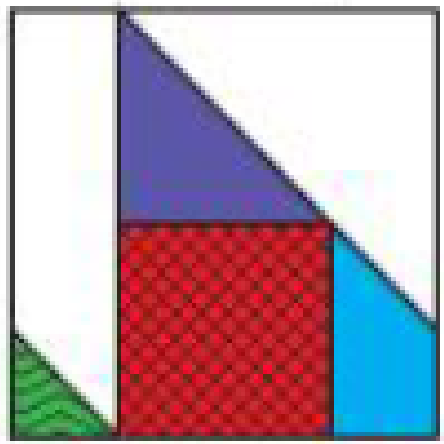
Example: Translation of a Colour Image



Original Image



Translated by 1/2 Square



Translated by 1/4 Square

- The translation is applied identically to each colour channel (R, G, B) represented by separate WFAs.

Extending to 3D Objects

Extending to 3D : Motivation

Motivation

Can we extend the success of using automata for 2D image representation and compression to 3D objects?

Extending to Three Dimensions

- Consider a 3D space, typically a $2^n \times 2^n \times 2^n$ grid of **voxels**.
- Each voxel has coordinates (x, y, z).
- **Alphabet Extension:** Use $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7\}$ corresponding to the 8 octants of a cube subdivision.
- **Address Encoding:** Interleave the bits of the coordinates:

$$a_i = 4x_i + 2y_i + z_i$$

  where x_i, y_i, z_i are the i-th bits of the x, y, z coordinates.
- **Representation (FSA):** A standard Finite State Automaton (FSA) can accept the set of address strings $w \in \Sigma^n$ corresponding to the "filled" voxels of the object.
- *(Note: WFAs could represent density, color, etc., but we focus on FSAs for basic shape representation here).*

3D Addressing

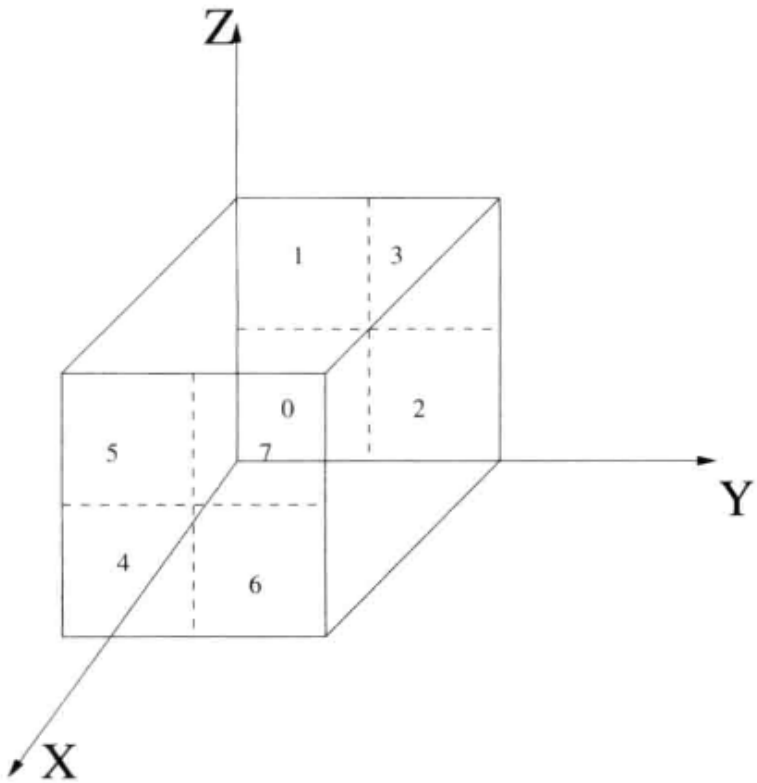
Octant Addressing

The alphabet $\Sigma = \{0..7\}$ maps to the 8 sub-cubes (octants). A string $w \in \Sigma^n$ specifies a unique voxel by repeatedly selecting octants based on the symbols in w.

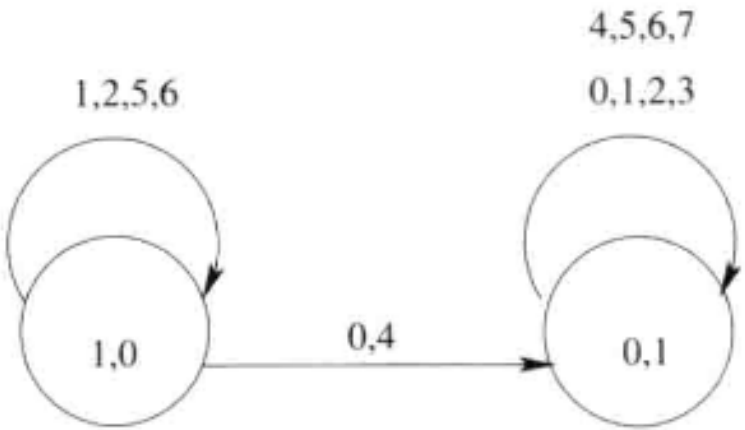
3D Addressing

Octant Addressing

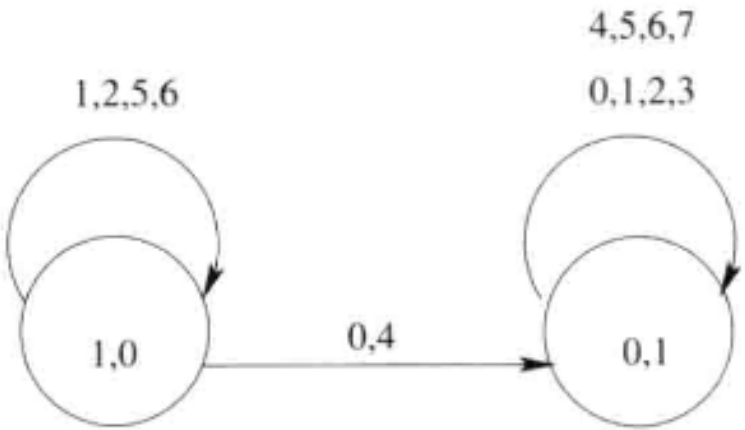
The alphabet $\Sigma = \{0..7\}$ maps to the 8 sub-cubes (octants). A string $w \in \Sigma^n$ specifies a unique voxel by repeatedly selecting octants based on the symbols in w.

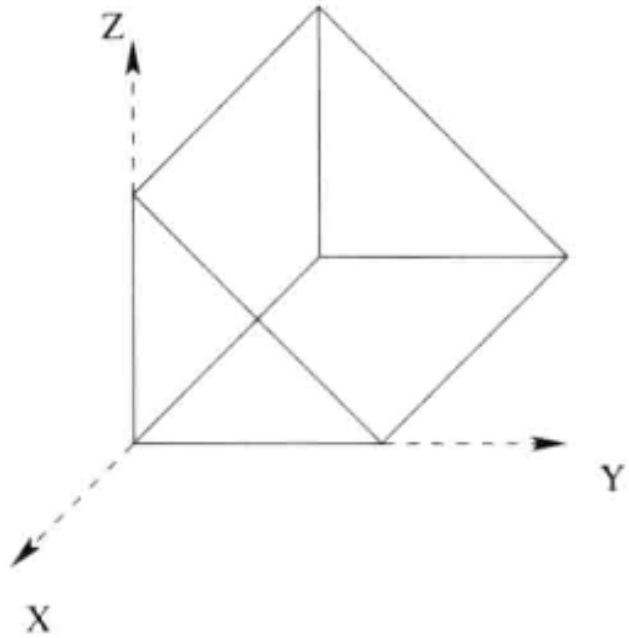


Example FSA and Corresponding 3D Shape



Example FSA and Corresponding 3D Shape





Transformations on 3D Objects

- Similar to 2D, transformations can be performed directly on the automata representation.
- Since the basic representation here is an FSA (accepting/rejecting voxels), we primarily use **Finite State Transducers (FSTs)**.
- These FSTs operate on the 3D alphabet $\Sigma = \{0, 1, ..., 7\}$.
- Transformations include scaling, translation, and rotation.

3D Scaling

Scaling by Factor 2 (Up) - Construction 20.4.1

Idea: Map one transition ’a’ in the new automaton to a path of length 2 (e.g., ’03’ for input ’0’) in the original automaton, effectively expanding the object.

Scaling by Factor 1/2 (Down) - Construction 20.4.2

Idea: Replace transitions ’a’ from the initial state(s) in the original automaton with paths of length 2 (e.g., ’0’ becomes path ’03’) using new intermediate states. This effectively shrinks the object.

3D Scaling Example

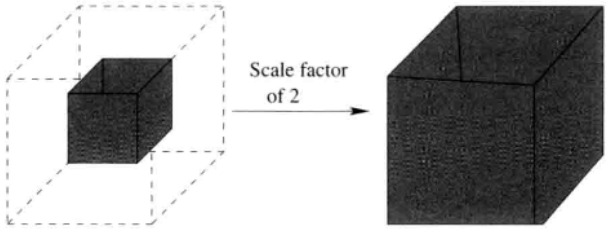


Figure 5: Scaling up by a factor of 2.

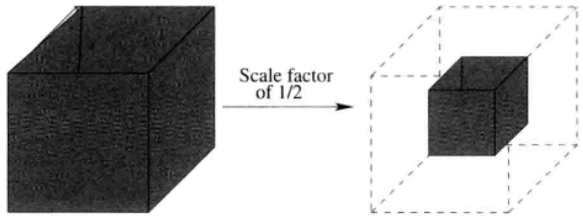


Figure 6: Scaling down by a factor of 1/2.

3D Scaling: FSA Constructions

Let the original FSA be $M = (\Sigma, Q, \delta, I, F)$.

Scaling by 2 (Up)

Construct $M' = (\Sigma, Q', \delta', I', F')$:

- Add a new initial state: $Q' = Q \cup \{q_0'\}$, $I' = \{q_0'\}$. $F' = F$.
- Define transitions from q_0': For each $a \in \Sigma$, $\delta'(q_0', a)$ includes all states reachable in M by a specific 2-symbol path p_a starting from an initial state in I. (e.g., if $a = 0$, path $p_0 = 03$. $\delta'(q_0', 0) = \delta(\delta(q_0, 0), 3)$).
- Original transitions remain: $\delta'(q, a) = \delta(q, a)$ for $q \in Q$.

3D Scaling: FSA Constructions

Scaling by 1/2 (Down)

Construct $M' = (\Sigma, Q', \delta', I', F')$:

- Add 8 new intermediate states: $Q' = Q \cup \{q'_0..q'_7\} \cup \{q_{new\_init}\}$, $I' = \{q_{new\_init}\}$. $F' = F$.
- Transitions from q_{new_init} go to intermediate states: $\delta'(q_{new\_init}, a) = \{q'_a\}$.
- Transitions from intermediate states simulate paths: $\delta'(q'_a, b)$ goes to states $q \in Q$ if q was reachable by path ab from an initial state in M. (e.g., $\delta'(q'_0, 3) = \{q \mid \exists i \in I, q = \delta(\delta(i, 0), 3)\}$).
- Original transitions remain within Q.

3D Translation - By bits

Method

Use FSTs over $\Sigma = \{0..7\}$ to modify address bits for specific coordinates.

Example: Translation along y-axis by +1 unit (with wrap-around):

- Requires changing the y-bit in $a_i = 4x_i + 2y_i + z_i$.
- $y_i = 0 \rightarrow y_i = 1$: Change a_i by +2 (e.g., $0 \rightarrow 2$, $1 \rightarrow 3$, $4 \rightarrow 6$, $5 \rightarrow 7$). FST has transitions like ‘0/2‘, ‘1/3‘, ‘4/6‘, ‘5/7‘.
- $y_i = 1 \rightarrow y_i = 0$: Change a_i by -2 (e.g., $2 \rightarrow 0$, $3 \rightarrow 1$, $6 \rightarrow 4$, $7 \rightarrow 5$). FST has transitions like ‘2/0‘, ‘3/1‘, ‘6/4‘, ‘7/5‘.

3D Translation - By bits

Method

Use FSTs over $\Sigma = \{0..7\}$ to modify address bits for specific coordinates.

Example: Translation along y-axis by +1 unit (with wrap-around):

- Requires changing the y-bit in $a_i = 4x_i + 2y_i + z_i$.
- $y_i = 0 \rightarrow y_i = 1$: Change a_i by +2 (e.g., $0 \rightarrow 2$, $1 \rightarrow 3$, $4 \rightarrow 6$, $5 \rightarrow 7$). FST has transitions like '0/2', '1/3', '4/6', '5/7'.
- $y_i = 1 \rightarrow y_i = 0$: Change a_i by -2 (e.g., $2 \rightarrow 0$, $3 \rightarrow 1$, $6 \rightarrow 4$, $7 \rightarrow 5$). FST has transitions like '2/0', '3/1', '6/4', '7/5'.
- Translation by 2, 4, ... units involves FSTs modifying higher-order bits.

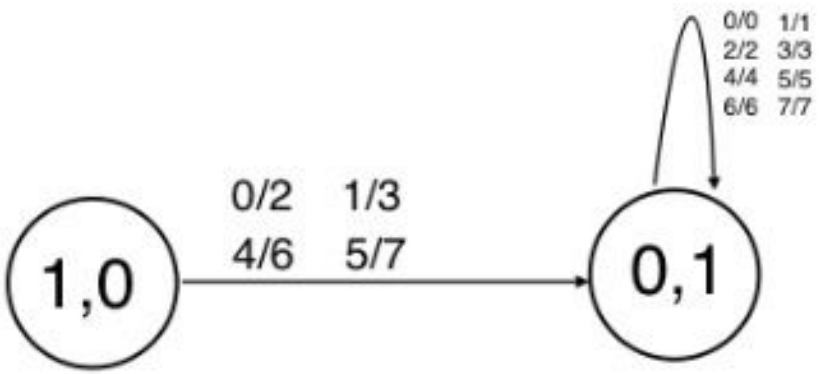
3D Translation - By size

Method

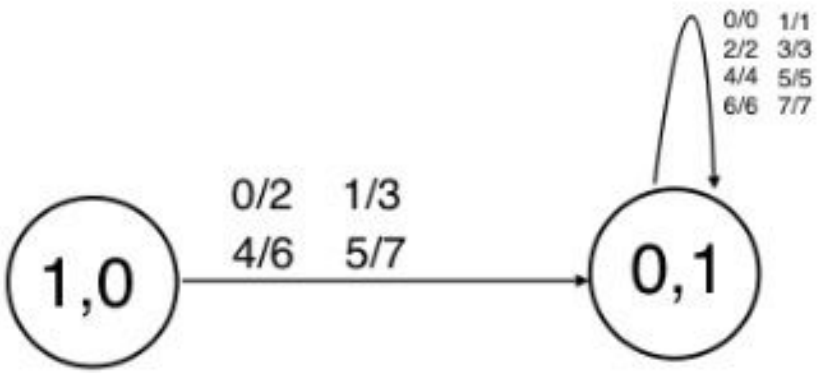
FSTs can also perform translations by large amounts or fractions of the object space size.

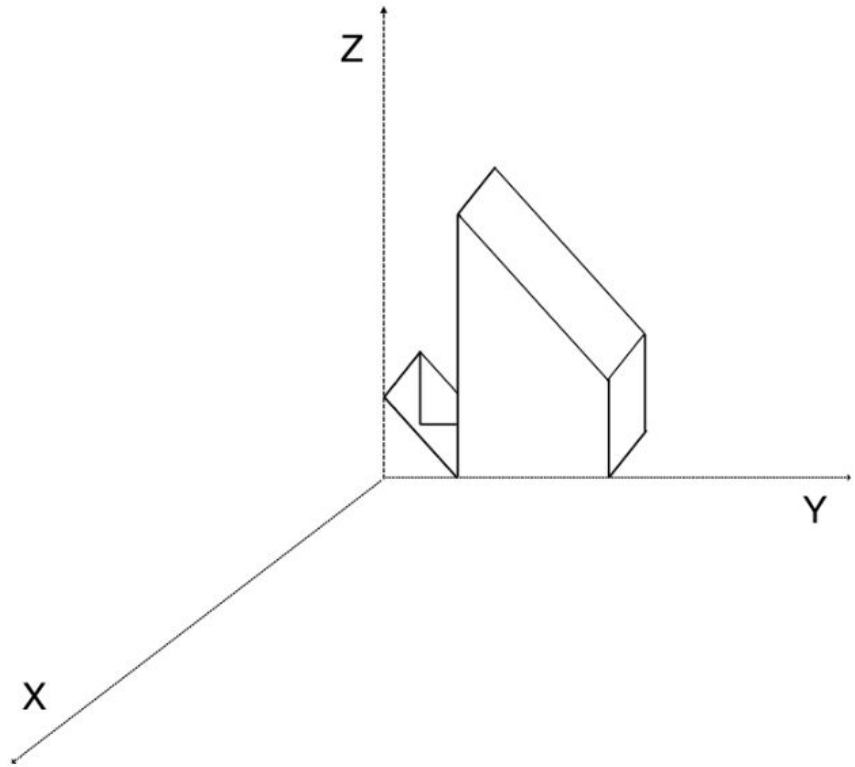
- **Translation by 1/2 Size:** Modify the most significant bit (MSB) of a coordinate (e.g., toggle y_{n-1}). This effectively shifts the object between the two halves of the space along that axis.
- **Translation by 1/4 Size:** Modify the two most significant bits.
- The FSTs directly manipulate the address symbols based on the required bit changes.

FST for Translation by 1/2 Size

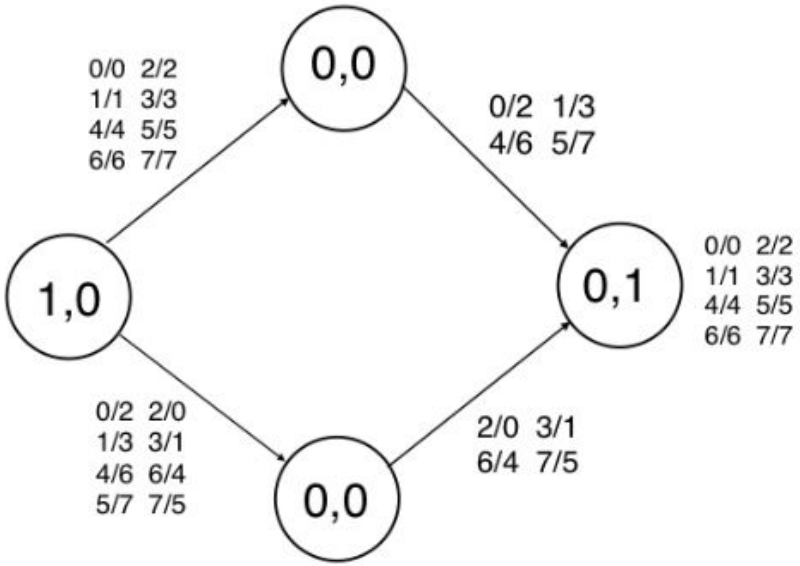


FST for Translation by 1/2 Size

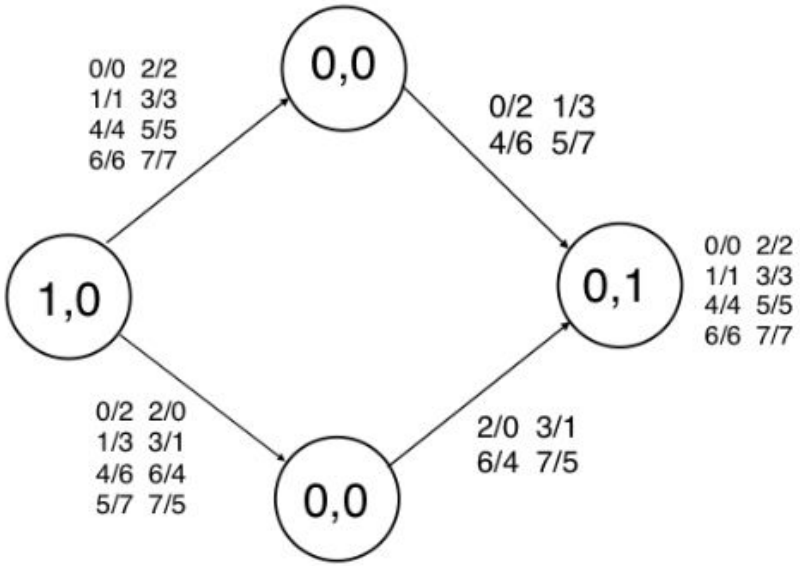


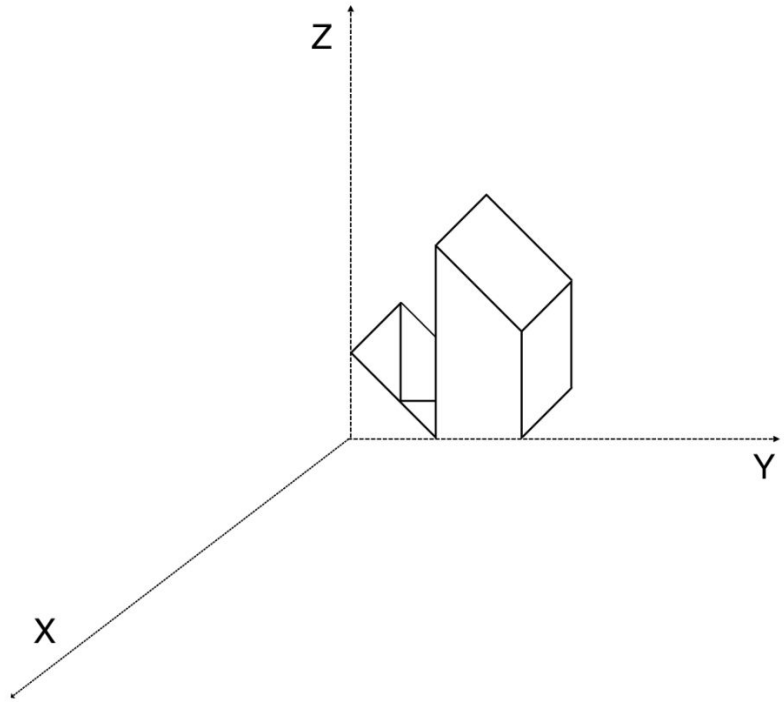


FST for Translation by 1/4 Size



FST for Translation by 1/4 Size





3D Rotation

Concept

Rotation of a 3D object (represented by FSA/WFA) around a coordinate axis (x, y, or z) is possible.

- The principle is similar to 2D rotation:
  1. Shift object center to the origin.
  2. Apply coordinate transformations using appropriate (likely more complex) WFTs for 3D rotation matrices.
  3. Shift object back.
- The specific WFT constructions for 3D rotation are more involved but are similar to 2D

Applications of FSA/WFA Image Representation

Application: Beyond Compression

The Automaton as Representation

Once an image is represented by a Finite Automaton (FSA for B&W, WFA for gray-scale), the automaton itself captures the image structure. This enables several applications beyond simple storage:

Application: Beyond Compression

The Automaton as Representation

Once an image is represented by a Finite Automaton (FSA for B&W, WFA for gray-scale), the automaton itself captures the image structure. This enables several applications beyond simple storage:

- **Image Reconstruction:** Generating pixel data back from the automaton at desired resolutions.
- **Interest Point / Feature Analysis:** The structure of the minimal automaton can highlight significant areas like corners or edges.
- **Similarity Recognition:** Comparing automata might provide a way to measure image similarity.

Application: Beyond Compression

The Automaton as Representation

Once an image is represented by a Finite Automaton (FSA for B&W, WFA for gray-scale), the automaton itself captures the image structure. This enables several applications beyond simple storage:

- **Image Reconstruction:** Generating pixel data back from the automaton at desired resolutions.
- **Interest Point / Feature Analysis:** The structure of the minimal automaton can highlight significant areas like corners or edges.
- **Similarity Recognition:** Comparing automata might provide a way to measure image similarity.

Focus

We will look primarily at Reconstruction and the link to Interest Points based on some examples.

Application: Interest Points & Automata Structure

Hypothesis: Automaton States $\leftrightarrow$ Image Features

The structure of the (minimal) automaton constructed from an image might inherently reflect visually important regions.

Application: Interest Points & Automata Structure

Hypothesis: Automaton States $\leftrightarrow$ Image Features

The structure of the (minimal) automaton constructed from an image might inherently reflect visually important regions.

- **Corner Detection Problem:** Identifying points where object boundaries change direction abruptly (L-corners, T-corners, Y-corners, X-corners) is crucial for image analysis.
- **Automaton Complexity:** Areas with fine details, textures, or sharp corners often require:
  - More states in the automaton to distinguish between slightly different sub-images.
  - Longer paths (deeper recursion) to represent the details.
- **Potential Correlation:** States that are ”created later” during the automaton construction or states with high local complexity might correspond to visually salient interest points (like corners).

Example: Reconstruction & State Marking

Reconstructed Image
(Lossless, 3317 states)

Marked States (Lighter points)

Example: Reconstruction & State Marking

Reconstructed Image
(Lossless, 3317 states)

Marked States (Lighter points)

Example: Reconstruction & State Marking

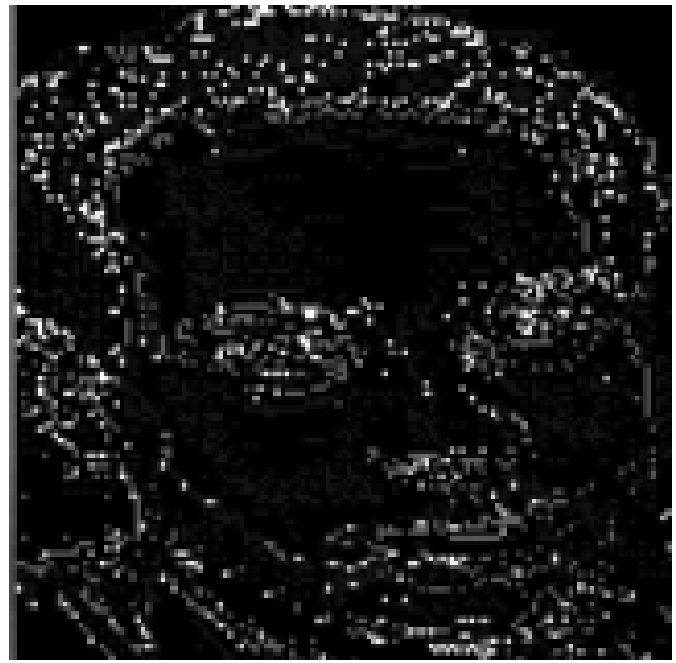
Reconstructed Image
(Lossless, 3317 states)

Marked States (Lighter points)

- The above figures shows a reconstruction from a large automaton (3317 states) achieving lossless compression.

Example: Reconstruction & State Marking

Reconstructed Image
(Lossless, 3317 states)



Marked States (Lighter points)

- The above figures shows a reconstruction from a large automaton (3317 states) achieving lossless compression.
- Some highlighted states correspond visually to interest points (eg: corners)

Methodology: Lossy Reconstruction & Feature Comparison

- **Lossy Automaton Construction**
- **Image Reconstruction (with Error)**
- **Standard Corner Detection**
- **Comparison**

Step 1: Lossy Automaton Construction

Goal: Compression via Approximation

Instead of aiming for a perfect (lossless) representation, construct a WFA that approximates the original image within a specified **error tolerance**.

Step 1: Lossy Automaton Construction

Goal: Compression via Approximation

Instead of aiming for a perfect (lossless) representation, construct a WFA that approximates the original image within a specified **error tolerance**.

Process Outcome

- During the automaton building process (like ”Construct Automaton for Recognition”), allow merging of states or transitions if the resulting difference in the represented sub-image is within the error threshold.
- This leads to an automaton with significantly **fewer states** compared to a lossless one (e.g., 175 states vs. 3317 in the following example).
- **Result:** A compressed representation, achieved by discarding some fine details or variations deemed acceptable by the error tolerance.

Step 2: Image Reconstruction (with Error)

Generating the Image

Use the smaller, lossy WFA (created in Step 1) as input to the image reconstruction procedure.

Step 2: Image Reconstruction (with Error)

Generating the Image

Use the smaller, lossy WFA (created in Step 1) as input to the image reconstruction procedure.

Characteristics of Reconstruction

- The reconstruction algorithm recursively traverses the automaton's states and transitions to assign gray values to pixels or regions.
- Since the automaton is lossy, the reconstructed image will **differ** from the original.
- The process might also involve **quantization**, reducing the number of possible output gray levels (e.g., to 16 levels in the following example), further contributing to compression.

Step 3: Standard Feature Detection (Edges/Corners)

Finding Important Pixels

Before complex analysis, images are often processed to find ”interesting” points like edges or corners, which represent significant changes in intensity or structure.

Step 3: Standard Feature Detection (Edges/Corners)

Finding Important Pixels

Before complex analysis, images are often processed to find ”interesting” points like edges or corners, which represent significant changes in intensity or structure.

Kernel-Based Methods (DL Approach)

A common technique involves sliding a small matrix (kernel or filter) over the image:

- The kernel performs a weighted sum of pixel neighbors (convolution).
- Different kernels highlight different features.
- Example: Edge Detection Kernel:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

- This kernel effectively zeros the surroundings where there is no edge thus highlighting the important features.

Step 4: Comparison

The Goal: Relating Automata Structure to Features

To test the hypothesis that the automaton structure reflects visual features, compare the locations marked based on the automaton states with features found using independent, standard methods.

Step 4: Comparison

The Goal: Relating Automata Structure to Features

To test the hypothesis that the automaton structure reflects visual features, compare the locations marked based on the automaton states with features found using independent, standard methods.

Process

- Take the reconstructed image.
- Overlay the locations corresponding to the marked automaton states (white dots).
- Overlay the locations identified by a standard corner/edge detector (red dots).
- Visually assess the degree of alignment or correlation between the two sets of points.

Step 4: Comparison

The Goal: Relating Automata Structure to Features

To test the hypothesis that the automaton structure reflects visual features, compare the locations marked based on the automaton states with features found using independent, standard methods.

Process

- Take the reconstructed image.
- Overlay the locations corresponding to the marked automaton states (white dots).
- Overlay the locations identified by a standard corner/edge detector (red dots).
- Visually assess the degree of alignment or correlation between the two sets of points.

Expected Outcome:

A noticeable overlap between the automaton-derived points and the standard feature points would support the idea that WFA structure analysis can be a meaningful way to identify image interest points.

Example: Error, Reconstruction & Corners

Original Image Ref.

Decompressed (16% error, 16 gray levels, 175 states)

Marked Corners (Red dots) + State dots (White)

Example: Error, Reconstruction & Corners

Original Image Ref.

Decompressed (16% error, 16 gray levels, 175 states)

Marked Corners (Red dots) + State dots (White)

Example: Error, Reconstruction & Corners

Original Image Ref.

Decompressed (16% error, 16 gray levels, 175 states)

Marked Corners (Red dots) + State dots (White)

Example: Error, Reconstruction & Corners

Original Image Ref.

Decompressed (16% error, 16 gray levels, 175 states)

Marked Corners (Red dots) + State dots (White)

- Middle: Reconstruction from a smaller automaton (175 states) with 16% error and fewer gray levels. Note blocky artifacts. White dots mark states ”without sub square”.

Example: Error, Reconstruction & Corners

Original Image Ref.

Decompressed (16% error, 16 gray levels, 175 states)

Marked Corners (Red dots) + State dots (White)

- Middle: Reconstruction from a smaller automaton (175 states) with 16% error and fewer gray levels. Note blocky artifacts. White dots mark states ”without sub square”.
- Right: Compares standard corner detection (red dots) with automaton state markers (white dots).

Example: Error, Reconstruction & Corners

Original Image Ref.

Decompressed (16% error, 16 gray levels, 175 states)

Marked Corners (Red dots) + State dots (White)

- Middle: Reconstruction from a smaller automaton (175 states) with 16% error and fewer gray levels. Note blocky artifacts. White dots mark states ”without sub square”.
- Right: Compares standard corner detection (red dots) with automaton state markers (white dots).
- **Key Observation:** Many detected corners (red) align with highlighted states (white), supporting the hypothesis.