RAIRO-Theor. Inf. Appl. **42** (2008) 217–236
DOI: 10.1051/ita:2007036

Available online at:
www.rairo-ita.org

BINARY OPERATIONS ON AUTOMATIC FUNCTIONS*

Juhani Karhumäki[1], Jarkko Kari[1] and Joachim Kupke[2]

Abstract. Real functions on the domain $[0,1)^n$ – often used to describe digital images – allow for different well-known types of binary operations. In this note, we recapitulate how weighted finite automata can be used in order to represent those functions and how certain binary operations are reflected in the theory of these automata. Different types of products of automata are employed, including the seldomly-used full Cartesian product. We show, however, the infeasibility of functional composition; simple examples yield that the class of automatic functions (*i.e.*, functions computable by automata) is not closed under this operation.

Mathematics Subject Classification. 68Q45, 68Q10, 68U10.

1. Introduction

The use of finite automata for image representation (and, ideally, compression and even manipulation) was proposed in [1] and [3]. Later, the model of weighted finite automata and their basic properties were discussed in [5]. Theoretical aspects were further studied in [7,10], and [9] while practical importance was discussed, *e.g.*, in [6]. One important feature of the theory is that many transformations of images can be performed directly on their compressed automata representation, see, *e.g.*, [4] and [8]. Some complexity-theoretic aspects were discussed in [11].

The goal of this note is to discuss – in the spirit of a unified survey – different ways of composing images, and how these operations can be carried out on the level of their automata representations.

Keywords and phrases. Automatic functions, weighted finite automata, full Cartesian product.

* *Research supported by the Academy of Finland grant 54102 and by the Swiss National Fund grant 200021-107327/1.*

[1] University of Turku, Finland; karhumak@cs.utu.fi, jkari@utu.fi

[2] ETH Zurich, Switzerland; joachimk@google.com

Article published by EDP Sciences

We consider three essentially different ways of composing digital images. First, the value of the composed image is defined pointwise, either by taking the sum or the product of the corresponding pixels of the images. Provided pixel values are nothing but gray values, these operations permit very natural interpretations. Another possibility to define the value of a pixel of the composed image is to use convolution. In all these cases, computing the desired composition is feasible in terms of standard operations on the automata representing the source images.

The second way of composing two pictures is to combine each pixel of the first image with each of the second one. More precisely, since pictures do not necessarily consist of discrete pixels in our context, out of an n_1-dimensional function $f_1 \colon [0,1)^{n_1} \to \mathbb{R}$ and an n_2-dimensional function $f_2 \colon [0,1)^{n_2} \to \mathbb{R}$, we compute the (n_1+n_2)-dimensional function $f \colon [0,1)^{n_1+n_2} \to \mathbb{R}$, defined by

$$f(u,v) = f_1(u) \oplus f_2(v) \quad \text{for some} \quad \oplus \in \{+,\cdot\}.$$

Interestingly, there is a natural, but not often used, operation on automata which realizes this composition if $\oplus = \cdot$, namely the full Cartesian product of automata. The term *full* accounts for taking the Cartesian product also of the input alphabets.

The third type of composition of images is obtained by considering these as functions into their domain and then taking functional composition. Although this is a very natural operation, the images realized by weighted finite automata are not closed under this operation, and we will give simple counterexamples.

2. Preliminaries

Definition 2.1 (weighted finite automata)**.** The quintuple $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ is called a *weighted finite automaton* iff:

- Q is a finite, non-empty set of states;
- Σ is a non-empty, finite alphabet;
- $I \colon Q \to \mathbb{R}$, the so-called *initial distribution,* is any function;
- $\Delta \colon Q \times \Sigma \times Q \to \mathbb{R}$, the so-called *transition distribution,* is any function;
- $F \colon Q \to \mathbb{R}$, the so-called *final distribution,* is any function.

For the reader familiar with non-weighted versions of finite automata, the above is a straightforward weighted generalization of nondeterministic automata. In fact, restricting I, Δ, and F to the range of $\{0,1\}$ instead of $\mathbb{R}$ yields exactly the definition of nondeterministic finite automata (with multiple initial states).

Definition 2.2. For every $\sigma \in \Sigma$, let $\Delta_\sigma \colon Q \times Q \to \mathbb{R}$ be defined by $\Delta_\sigma(q,q') := \Delta(q,\sigma,q')$. We will think of Δ_σ as a matrix. For a word $w \in \Sigma^*$, we inductively define Δ_w by

$$\begin{aligned} \Delta_\varepsilon &:= \mathbb{1}_Q, \\ \Delta_{\sigma v} &:= \Delta_\sigma \cdot \Delta_v \quad \text{for} \quad \sigma \in \Sigma, v \in \Sigma^*, \end{aligned}$$

where $\mathbb{1}_Q$ is the $Q \times Q$ identity matrix, *i.e.*,

$$\mathbb{1}_Q(q, q') := \begin{cases} 1 & \text{if } q = q'; \\ 0 & \text{otherwise.} \end{cases}$$

If we interpret I and F as (column) vectors of dimension $|Q|$, the output of $\mathcal{A}$ on a word $w \in \Sigma^*$, denoted by $\mathcal{A}(w)$, is just the real value $I^{tr} \cdot \Delta_w \cdot F$. Note that this, again, generalizes the concept of nondeterministic automata where outputs in our sense are always natural numbers and where a word is said to be recognized by an automaton iff its output amounts to a non-zero value. (In fact, what we call "output" here is called "ambiguity" in [13,14].) Accordingly, we have the following, rather intuitive interpretation of what is a computation of a weighted finite automaton: for every path with a labeling compatible with the input word, compute the product of the weights of its edges. Then, compute the sum of these values over all such paths.

We will use alphabets whose symbols correspond to partitions of a given region of space points. A word over such an alphabet corresponds, in a recursive sense, to a partition (determined by its last symbol) of a partition (determined by the word minus its last symbol) – a more precise definition will follow. Consequently, as words become longer, they correspond to ever-smaller regions, and an infinite word corresponds to a single point, in each of which, ultimately, a function needs to be defined. This is why we will need the following definition.

Definition 2.3. Let, as usual, Σ^ω be the set of *infinite* words over Σ. We extend the domain of $\mathcal{A}$, regarded as a function, from Σ^* to $\Sigma^* \uplus \Sigma^\omega$ by

$$\mathcal{A}(z) := \lim_{k \to \infty} \mathcal{A}(\mathrm{pref}_k(z))$$

for all $w \in \Sigma^\omega$ where $\mathrm{pref}_k(z)$ denotes the first k symbols of z. Note that this limit need not always exist, and thus the function $\mathcal{A}$ may be undefined for some infinite words.

In what follows, we will define a subclass of weighted finite automata in order to guarantee the existence of the limit in Definition 2.3 for all $z \in \Sigma^\omega$. We can think of the automata from this class as acyclic weighted finite automata, plus, potentially, loops from some states to themselves, which are weighted by some value less than 1 or, if such a loop is the only transition leaving some state, its weight will be exactly 1.

Formally, we will use the concept of the *spectral radius* of a matrix in Definition 2.4. Recall that the spectral radius of a matrix M is

$$\varrho(M) := \sup\{|\lambda| \mid \text{for some vector } v,\ M \cdot v = v \cdot \lambda\},$$

i.e., the maximum ratio by which eigenvectors of M may be stretched. In contrast, for any vector norm $\|\cdot\|$, the induced matrix norm is

$$\|M\| := \sup\{\lambda \mid \text{for some vector } v,\ \|M \cdot v\| = \|v\| \cdot \lambda\},$$

i.e., the maximum factor by which the norm of any vector may be multiplied as a result of applying M to it.

Recall that the spectral radius of a triangular matrix is just the maximum absolute value of its diagonal entries, and recall that by GELFAND's formula, for any vector norm $\|\cdot\|$ and any matrix A, the induced norm of its powers converges to the respective power of its spectral radius, *i.e.*,

$$\varrho(A) = \lim_{k\to\infty} \|A^k\|^{1/k}. \tag{1}$$

Our restrictions may seem grave at first glance, but the resulting class of automata is expressive enough to compute a wide class of functions. Ongoing research on the *joint spectral radius* (*e.g.*, [2]) aims at settling the question to what extent our restrictions might be relaxed.

Definition 2.4. Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a weighted finite automaton. We say that $\mathcal{A}$ is a *level automaton* iff

(i) for all $\sigma \in \Sigma$, the matrix Δ_σ is upper-triangular with non-negative entries only;

(ii) there is a number $\ell \in \mathbb{N}$ such that all the matrices Δ_σ have the form

$$\Delta_\sigma = \left(\begin{array}{c|c} A_\sigma & B_\sigma \\ \hline 0 & \mathbb{1}_\ell \end{array}\right),$$

where the spectral radius of A_σ is less than one, *i.e.*,

$$\varrho(A_\sigma) < 1 \quad \text{; and}$$

(iii) the final distribution is non-negative, *i.e.*, $F(q) \geq 0$ for all $q \in Q$.

The last restriction serves simplification purposes and is not essential; any weighted finite automaton can easily be turned into one that satisfies condition (iii). Simply enough, we can have a copy of a given automaton compute the same function, but with initial and final weights negated. By canceling all negatively weighted final weights, it is then possible to reassemble the originally computed function. This fact is summarized in Observation 2.5.

Observation 2.5. Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a weighted finite automaton. Define the weighted finite automaton $\mathcal{A}_+$ by

$$\begin{aligned} \mathcal{A}_+ &:= (Q \times \{1, -1\}, \Sigma, I_+, \Delta_+, F_+) \quad \text{where} \\ I_+(q, b) &:= b \cdot I(q), \\ F_+(q, b) &:= (b \cdot F(q) + |F(q)|)/2 \quad \text{, and} \\ \Delta_+((q, b), \sigma, (q', b')) &:= |\tfrac{b+b'}{2}| \cdot \Delta(q, \sigma, q'). \end{aligned}$$

Obviously, $\mathcal{A}$ and $\mathcal{A}_+$ compute the same function, and $\mathcal{A}_+$ satisfies condition (iii) of Definition 2.4.

Lemma 2.6. *Level automata $\mathcal{A}$, seen as partial functions from Σ^ω, are in fact complete functions (defined everywhere).*

Proof. Let $w \in \Sigma^*$ be any word, and define A_w analogously to Δ_w. It is easy to see that

$$\Delta_w = \left(\begin{array}{c|c} A_w & B_w \\ \hline 0 & \mathbb{1}_\ell \end{array}\right), \tag{2}$$

where B_ε is the (appropriately-dimensioned) zero matrix and

$$B_{\underbrace{w_1 \ldots w_k}_{=:w}} := \sum_{j=0}^{k-1} A_{w_1 \ldots w_j} B_{w_{j+1}} = \sum_{\sigma \in \Sigma} \Bigg(\sum_{\substack{x \in \Sigma^* \\ \exists y \in \Sigma^* \colon x\sigma y = w}} A_x \Bigg) \cdot B_\sigma \tag{3}$$

for all $k \geq 1$. First of all, we will prove that for all infinite words $z \in \Sigma^\omega$, the infinite sum

$$\sum_{j=0}^{\infty} A_{\mathrm{pref}_j(z)} \tag{4}$$

converges, which implies that the matrices A_w tend to the zero matrix as $|w|$ tends to infinity. To this end, let A be the entry-wise maximum matrix of the matrices A_σ. Naturally, A is upper-triangular and has spectral radius $\varrho(A) < 1$. (Note that this would not necessarily hold had we not required the matrices A_σ to be triangular.) Likewise, it is straightforward to see that

$$\|A_w\| \leq \|A^{|w|}\| \tag{5}$$

for an arbitrary vector norm $\|\cdot\|$ (because A_w is entry-wise less-or-equal $A^{|w|}$). By (1), we have

$$\|A^k\|^{1/k} \leq \varrho(A) \qquad \text{for sufficiently large } k$$

and hence

$$\|A_w\|^{1/|w|} \leq \varrho(A) \qquad \text{for sufficiently long words } w, \tag{6}$$

which means that by the root test for infinite sums [12],

$$\limsup_{j \to \infty} \sqrt[j]{\|A_{\mathrm{pref}_j(z)}\|} \leq \varrho(A) < 1, \tag{7}$$

we know that the infinite sum (4) converges (absolutely).

Consequently, also those sums

$$\sum_{\substack{x \in \Sigma^* \\ \exists y \in \Sigma^* \colon x\sigma y = w}} A_x$$

in (3) which are infinite converge because they are subsums of (4). Thus, the upper-right part of the matrices Δ_w in (2) converges to some matrix as $|w|$ approaches infinity. At the same time, the upper-left part of the matrices Δ_w vanishes. □

As a consequence, the functions $\mathcal{A}\colon \Sigma^\omega \to \mathbb{R}$ induced by level automata are not only defined everywhere, they are also continuous in all $w \in \Sigma^\omega$, where "continuous" in an infinite word $w \in \Sigma^\omega$ means that for all $\varepsilon > 0$ there exists $k \in \mathbb{N}$ such that $|\mathcal{A}(\mathrm{pref}_k(w)w') - \mathcal{A}(w)| < \varepsilon$ for all $w' \in \Sigma^\omega$.

Observation 2.7. Level automata $\mathcal{A}$, seen as functions from Σ^ω, are continuous everywhere.

Proof. Let $w \in \Sigma^*$ be a finite word, and let $z, z' \in \Sigma^\omega$ be infinite words. The idea is to prove that $|\mathcal{A}(wz) - \mathcal{A}(wz')|$ becomes arbitrarily small for sufficiently long words w, which implies that $\mathcal{A}$ is continuous in wz, which implies that $\mathcal{A}$ is continuous everywhere. Since the upper-left part of the matrices Δ_w in (2) tends to the zero matrix, it is sufficient to prove that the upper-right parts, matrices B_w, remain close to one another. But we have practically already seen this, since

$$\begin{aligned}
\left\| \lim_{j\to\infty} B_{\mathrm{pref}_j(wz)} - \lim_{j\to\infty} B_{\mathrm{pref}_j(wz')} \right\| &= \left\| \lim_{j\to\infty} (B_{\mathrm{pref}_j(wz)} - B_{\mathrm{pref}_j(wz')}) \right\| \\
&\overset{(3)}{=} \left\| \sum_{j=0}^{\infty} (A_{\mathrm{pref}_j(wz)} B_{(wz)_{j+1}} - A_{\mathrm{pref}_j(wz')} B_{(wz')_{j+1}}) \right\| \\
&\overset{(\star)}{=} \left\| \sum_{j=|w|}^{\infty} (A_{\mathrm{pref}_j(wz)} B_{(wz)_{j+1}} - A_{\mathrm{pref}_j(wz')} B_{(wz')_{j+1}}) \right\| \\
&= \left\| A_w \cdot \sum_{j=0}^{\infty} (A_{\mathrm{pref}_j(z)} B_{z_{j+1}} - A_{\mathrm{pref}_j(z')} B_{z'_{j+1}}) \right\| \\
&\leq \|A_w\| \cdot \sum_{j=0}^{\infty} \left[\|A_{\mathrm{pref}_j(z)}\| \cdot \|B_{z_{j+1}}\| + \|A_{\mathrm{pref}_j(z')}\| \cdot \|B_{z'_{j+1}}\| \right] \\
&\overset{(5)}{\leq} \|A_w\| \cdot \max_{\sigma\in\Sigma} \|B_\sigma\| \cdot 2 \cdot \sum_{j=0}^{\infty} \|A^j\|,
\end{aligned}$$

where we have equality in $(\star)$ because for $j \in \{0, \ldots, |w| - 1\}$, the $(j+1)$-prefixes of wz and wz' match, and thus the term $A_{\mathrm{pref}_j(wz)} B_{(wz)_{j+1}} - A_{\mathrm{pref}_j(wz')} B_{(wz')_{j+1}}$ vanishes.

By (6), we know that on the one hand, $\|A_w\|$ becomes arbitrarily small for sufficiently long words w. On the other hand, we know by (7) that the infinite sum

$$\sum_{j=0}^{\infty} \|A^j\|$$

converges, and obviously, the value it converges to is not dependent on z or z'. □

Observation 2.8. Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a level automaton. If we are only interested in the function $\mathcal{A}\colon \Sigma^\omega \to \mathbb{R}$, we may assume, w. l. o. g., that $F(q) = 1$ for all $q \in Q$.

Proof. Let $P \subseteq Q$ be the ℓ-set of states such that $\Delta_\sigma|_{P\times P} = \mathbb{1}_\ell$ (for all $\sigma \in \Sigma$). By the last sentence of the proof of Lemma 2.6, the weights $F(q)$ are *irrelevant* for all states $q \in Q \setminus P$. Moreover, states $q \in P$ with $F(q) = 0$ are useless and can be removed (because they cannot be left and can thus not contribute to final weight). Now, for any $q \in P$ with $F(q) \neq 1$, we simply define

$$\begin{aligned} I'(q) &:= I(q) \cdot F(q), \\ \Delta'(q', q) &:= \Delta(q', q) \cdot F(q), \quad \text{and} \\ F'(q) &:= 1, \end{aligned}$$

which gives us the automaton $\mathcal{A}' = (Q, \Sigma, I', \Delta', F')$. On infinite words, $\mathcal{A}'$ computes the same values as $\mathcal{A}$, but on finite words, their outputs may differ since in $\mathcal{A}$, states from $Q \setminus P$ may have been assigned a final weight $\neq 1$. □

In what follows, we will restrict ourselves to the case of $\Sigma = \{0, 1\}^{n\times 1}$ for some $n \in \mathbb{N} \setminus \{0\}$, *i.e.*, our input alphabet consists of all binary columns of a fixed height n where n is the dimension of the domain of the functions of our predominant interest. In particular, Σ^ω is the set of all infinite-to-the-right matrices over $\{0, 1\}$ of height n.

Definition 2.9. Define $\Sigma^{\omega_0} \subset \Sigma^\omega$ to be the set of those matrices from Σ^ω where every row contains an infinite number of zeros. The reason for this provision is that we intend to view (finite and infinite) words as binary representations of real numbers from $[0, 1)$, and real numbers with finite representation are well-known to have another, infinite representation. (For example, the number $\frac{1}{2}$ has the two binary representations $0.1000\ldots_{(2)}$ and $0.0111\ldots_{(2)}$.) We would like to avoid any confusion that this fact may cause.

Definition 2.10 (Average-preserving property)**.** Let $\mathcal{A}$ be a weighted finite automaton. If, for every word $w \in \Sigma^*$, we have

$$\mathcal{A}(w) = \frac{1}{|\Sigma|} \sum_{\sigma\in\Sigma} \mathcal{A}(w\sigma),$$

then we say that $\mathcal{A}$ is *average-preserving.*

Preserving the average weight is important for our automata for two reasons. First, it is practical: If we want to produce the represented picture as a matrix of gray values at a given resolution, we simply feed the automaton in question with shorter or longer coordinate words, and this yields a sensibly shrunk or enlarged version of the picture in question, respectively.

Second, when it comes to computing integrals of the functions computed by finite automata, their average-preserving property is the essential building block

in ensuring that a finite automaton to compute the integral function can be constructed easily.

Observation 2.11. Every level automaton $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ can be assumed to be average-preserving, w. l. o. g.

Proof. We first note that by Observation 2.8, we may assume a constant final weight of 1. It is sufficient to guarantee that for all states $q \in Q$, we have

$$\sum_{\substack{q' \in Q \\ \sigma \in \Sigma}} \Delta(q, \sigma, q') = |\Sigma|. \tag{$\star$}$$

Call any state $q \in Q$ which satisfies condition ($\star$) average-preserving. Let P be defined as in Observation 2.8. All states in P are trivially average-preserving. For $q \in Q \setminus P$, we can compute the number

$$\alpha_q := \frac{|\Sigma| - \sum_{\sigma \in \Sigma} \Delta(q, \sigma, q)}{\sum_{\substack{q' \in Q \setminus \{q\} \\ \sigma \in \Sigma}} \Delta(q, \sigma, q')},$$

which is the ratio by which the weight of outgoing edges has to be adjusted so as to yield condition ($\star$). Like in Observation 2.8, if we multiply outgoing edges by α_q, we need to divide incoming ones, as well as the value $I(q)$, by $\alpha_q \neq 0$ in order to preserve the computed output.

This procedure, applied to a state q, potentially destroys the average-preserving property of in-neighbors of q, but does nothing to this property of out-neighbors of q. Hence, we can propagate the average-preserving property using breadth-first search from P to Q. □

From now on, we will be interested in average-preserving level automata only. Let us therefore abbreviate these as "finite automata".

Definition 2.12 (Words and space points)**.** For every (infinite) word $w \in \Sigma^{\omega_0}$, we define its *associated n-dimensional space point* $x(w) \in [0, 1)^n$ by

$$x(\sigma v) := \frac{1}{2}(\sigma + x(v)) \quad (\text{for } \sigma \in \Sigma,\ v \in \Sigma^{\omega_0}),$$

or, equivalently, for $w = \sigma_1\sigma_2\sigma_3\ldots$,

$$x(\sigma_1\sigma_2\sigma_3\ldots) := \sum_{i \geq 1} \sigma_i \cdot 2^{-i}.$$

We naturally extend the domain of x to $\Sigma^{\omega_0} \uplus \Sigma^*$ by padding finite words with infinitely many zero bits, *i.e.*,

$$x(w) := x(w((0, \ldots, 0)^{tr})^{\omega}) \text{ for all } w \in \Sigma^*.$$

Nevertheless, for any $y \in [0,1)^n$, we will use $x^{-1}(y)$ in order to denote the single infinite word $w \in \Sigma^{\omega_0}$ which satisfies $x(w) = y$.

Definition 2.13. We extend the domain of $\mathcal{A}$, regarded as a function, from $\Sigma^* \uplus \Sigma^\omega$ to $\Sigma^* \uplus \Sigma^\omega \uplus [0,1)^n$ by

$$\mathcal{A}(y) := \mathcal{A}(x^{-1}(y))$$

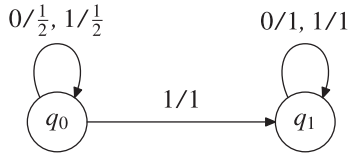
for every $y \in [0,1)^n$ whenever $\mathcal{A}$ is defined on $x^{-1}(y)$.

Definition 2.14 (Automatic functions)**.** Let $f\colon [0,1)^n \to \mathbb{R}$ be any function. Call the class of functions of this kind the class of *real functions.* Every (average-preserving weighted level) finite automaton computes a real function. Call the class of functions that can be computed by finite automata the class of *automatic functions.*

Example 2.15. Consider the one-dimensional *identity* function

$$f\colon [0,1) \to \mathbb{R}, \quad f(x) := x.$$

A finite automaton can compute it in the following natural way. Nondeterministically, find all bits which are set to one; weight every such bit according to its position; and finally, take the sum of all these weights. This gives us the following automaton.



$$I(q_0) = \frac{1}{2} \qquad I(q_1) = 0 \qquad F(q_0) = F(q_1) = 1.$$

Note that $F(q_0) = 1$ so as to make the automaton average-preserving. Indeed, adding the value $2^{-|w|}$ to the value $x(w)$ for finite words $w \in \{0,1\}^*$ yields the middle $x(w1)$ of the interval $[x(w), x(w1^\omega))$, which w represents.

For example, $\frac{3}{8} = 0.011_{(2)}$, but the word 011 is the common prefix of the binary representations of numbers from $[\frac{3}{8}, \frac{1}{2})$, and in fact, it does not only admit the paths

$$\underbrace{q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_1}_{\text{weight } \frac{1}{2}\cdot\frac{1}{4}\cdot 1 = \frac{1}{8}} \qquad \underbrace{q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_1}_{\text{weight } \frac{1}{2}\cdot\frac{1}{2}\cdot 1 = \frac{1}{4}}$$

in this automaton, the sum of whose weights is, indeed, $\frac{3}{8}$. But the path

$$\underbrace{q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0}_{\text{weight } \frac{1}{2}\cdot\frac{1}{8}\cdot 1 = \frac{1}{16}}$$

contributes another $\frac{1}{16}$, adding up to $\frac{7}{16}$, which is the average of (all the numbers from the interval delimited by) $\frac{3}{8}$ and $\frac{1}{2}$.

Unlike in the preceding example, automatic functions need not be continuous, but points of discontinuities can only occur when all coordinates have finite binary representations. The reason is that it is exactly (rational) numbers with finite binary representations which allow for two possible infinite binary representations, namely one with almost only zeros and one with almost only ones. We have formally forbidden the latter, but we can get arbitrarily close to, for instance, the value $\frac{1}{2} = 0.1000\ldots_{(2)}$ by the sequence

$$0.01000\ldots_{(2)}, 0.01100\ldots_{(2)}, 0.01110\ldots_{(2)}, \ldots$$

This fact can easily be exploited in order to construct finite automata computing functions with discontinuities.

Lemma 2.16. *An automatic function can be discontinuous only in points whose coordinates have finite binary representations (in some dimensions).*

Proof. Obvious from Observation 2.7. □

There is even more known about automatic functions. In particular, an automatic function is derivable everywhere and arbitrarily often iff it is a polynomial function. On the other hand, every polynomial function of degree d can be computed by an automaton with $\mathcal{O}(d)$ states. Hence, automatic functions can be either

- polynomial functions (in a natural way, *i.e.*, without investing a huge number of states);
- continuous functions that somewhere are only finitely often derivable; or
- functions with discontinuities at points with finite binary representations.

In practical applications such as signal processing (*e.g.*, image processing, with possible extensions to movie processing and/or sound processing), automatic functions and the underlying theory of weighted finite automata have been discovered to be excellent tools [6]. Hence, the reader should focus on situations where $n \leq 2$ and function values are interpreted as shades of gray. This is, however, no limit to our considerations, but merely an illustrative help.

In this note, we capitalize on binary operators on automatic functions. In fact, we would like to classify operators into four kinds, namely

(1) point-wise operations such as addition and multiplication;
(2) local but not point-wise operations;
(3) coordinate-wise operations that usually enlarge the value n;
(4) operations assuming same topology of domain and range – such as composition.

Typically, local, point- and coordinate-wise operations are feasible, *i.e.*, the class of automatic functions is closed under these operations, *and* there are efficient algorithms that compute an appropriate automaton out of automata representing the operands. On the other hand, and this amounts to our first observation, the domain and the range of automatic functions are inherently different, *i.e.*, even if

we restrict ourselves to automatic functions whose range is a subset of $[0,1)$, the resulting class of functions is not closed under composition.

3. Non-closedness of automatic functions under composition

Observation 3.1. *There are automatic functions*

$$f_1\colon [0,1) \to \mathbb{R} \quad \textit{and} \quad f_2\colon [0,1) \to \mathbb{R}$$

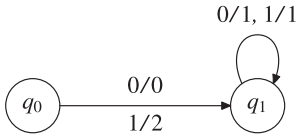
such that $f_1([0,1)) \subseteq [0,1)$ *and* $f_2 \circ f_1$ *(i.e., the function that maps* x *to* $f_2(f_1(x))$ *for all* x*) is not automatic.*

Proof. Let

$$f_2(x) := \begin{cases} 0 & \text{if } 0 \le x < \frac{1}{2}; \\ \frac{1}{2} & \text{otherwise.} \end{cases}$$

This function is clearly automatic since only the first digit of its argument (which represents whether it is $\ge \frac{1}{2}$) needs to be taken into account in order to decide whether its value is 0 or $\frac{1}{2}$.

The following automaton, consequently, computes f_2.



$$I(q_0) = \frac{1}{4} \qquad I(q_1) = 0 \qquad F(q_0) = F(q_1) = 1.$$

Note that reading the second through the last digit always yields a weight of 1. It is hence easy to see that the automaton computes no other function than f_2.

Now, let $f_1(x) := \frac{3}{4}x$. Since f_1 is a polynomial function, it is an automatic function. Obviously,

$$(f_2 \circ f_1)(x) = \begin{cases} 0 & \text{if } \quad 0 \le x < \frac{2}{3}; \\ \frac{1}{2} & \text{otherwise,} \end{cases}$$

and consequently, $f_2 \circ f_1$ is non-continuous in $x = \frac{2}{3}$. Since $\frac{2}{3} = 0.1010101010\ldots_{(2)}$, it has no finite binary representation, and so, by Lemma 2.16, $f_2 \circ f_1$ is not automatic.

Note that one might argue that by changing the arity of the input representation, this obstacle could be overcome. In fact, $\frac{2}{3} = 0.2_{(3)}$, and for any rational number q, there is an arity of input representation such that q may be represented finitely. But even for a possible extension of automaticity of functions,

the corresponding class of functions is not closed under composition. In order to see this, let

$$f_1(x) := x^2,$$

which is still automatic since it is a polynomial function. Obviously,

$$(f_2 \circ f_1)(x) = \begin{cases} 0 & \text{if } 0 \leq x < \frac{1}{2}\sqrt{2}; \\ \frac{1}{2} & \text{otherwise,} \end{cases}$$

and consequently, f_2 is non-continuous in $x = \frac{1}{2}\sqrt{2}$, which for every arity of input representation is non-periodic and therefore has only infinite representations. □

If we ask whether smooth automatic functions (*i.e.*, automatic functions that are derivable everywhere) are closed under composition, however, the answer is "yes," since the only smooth automatic functions are polynomial functions and all polynomial functions are automatic.

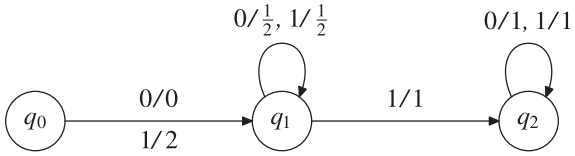
But it is natural to ask whether the class of continuous automatic functions is closed under composition. The answer is negative, again.

Observation 3.2. *There are continuous automatic functions* $f_1 \colon [0,1) \to \mathbb{R}$ *and* $f_2 \colon [0,1) \to \mathbb{R}$ *such that* $f_1([0,1)) \subseteq [0,1)$*, and* $f_2 \circ f_1$ *is not automatic.*

Proof. Let

$$f_2(x) := \begin{cases} 0 & \text{if } 0 \leq x < \frac{1}{2}; \\ x - \frac{1}{2} & \text{otherwise.} \end{cases}$$

This function is easily seen to be automatic if we combine the idea from Example 2.15 with that from the automaton for the function f_2 in Observation 3.1. More specifically, depending on the first bit of input, we should either output zero or compute the identity on the remainder of the input. This gives us the following automaton.



$$I(q_0) = \frac{1}{8} \qquad I(q_1) = I(q_2) = 0 \qquad F(q_0) = F(q_1) = F(q_2) = 1.$$

Again, let $f_1(x) := \frac{3}{4}x$. Obviously,

$$(f_2 \circ f_1)(x) = \begin{cases} 0 & \text{if } 0 \leq x < \frac{2}{3}; \\ \frac{3}{4}x - \frac{1}{2} & \text{otherwise.} \end{cases}$$

Of course, $f_2 \circ f_1$ is continuous even in $x = \frac{2}{3}$ since the composition of continuous functions yields, again, a continuous function. Suppose, however, $f_2 \circ f_1$ were automatic. By [4], we know that then, $(f_2 \circ f_1)'$, *i.e.*, its derivative, would also be automatic. But $(f_2 \circ f_1)'$ is non-continuous in $x = \frac{2}{3}$, which contradicts its automaticity.

Again, as in Observation 3.1, by replacing f_1 with $f_1(x) := x^2$, we deduce that even changing the arity of input representation does not remedy the non-closedness of automatic functions under composition. □

4. Additions

Point-wise addition is presumably the easiest operation on finite automata. Constructing an automaton to compute the sum of two automatic functions, given in terms of the automata computing them, is immediate: just take the (disjoint) union of the two sets of states and extend I, Δ, and F in the natural way.

For coordinate-wise operations, assume we have an n_1-dimensional automatic function f_1 and an n_2-dimensional function f_2. Now let $n := n_1 + n_2$, and, for some $\oplus \in \{+, \cdot\}$, define the n-dimensional function f by

$$f(x_1, \ldots, x_{n_1}, x_{n_1+1}, \ldots, x_n) := f_1(x_1, \ldots, x_{n_1}) \oplus f_2(x_{n_1+1}, \ldots, x_n). \tag{8}$$

In this section, $\oplus = +$, of course. In order to accomplish coordinate-wise addition, extend f_1 and f_2 to $[0,1)^n$ as appropriate. Do so by extending Δ_i, $i = 1, 2$, in the straightforward way: let

$$\widehat{\Delta_i}(q, \widehat{\sigma}, q') := \Delta_i(q, \sigma_i(\widehat{\sigma}), q'),$$

where σ_1 is the projection of $\widehat{\sigma}$ to the first n_1 and where σ_2 is the projection of $\widehat{\sigma}$ to the last n_2 coordinates. Secondly, perform point-wise addition.

Both of these procedures, if applied to level automata, yield level automata in turn. This also holds for the procedures in the next two sections.

5. Multiplications

Point-wise multiplication can be performed using a very well-known construction in automata theory, namely the *standard Cartesian product* of two automata $\mathcal{A}_i = (Q_i, \Sigma, I_i, \Delta_i, F_i)$, for $i \in \{1, 2\}$, which is

$$\mathcal{A} := \mathcal{A}_1 \otimes \mathcal{A}_2 := (Q_1 \times Q_2, \Sigma, I, \Delta, F)$$

where

$$\begin{aligned} I(q_1, q_2) &:= I_1(q_1) \cdot I_2(q_2), \\ \Delta((q_1, q_2), \sigma, (q_1', q_2')) &:= \Delta_1(q_1, \sigma, q_1') \cdot \Delta_2(q_2, \sigma, q_2'), \text{and} \\ F(q_1, q_2) &:= F_1(q_1) \cdot F_2(q_2). \end{aligned}$$

It is immediate that $\mathcal{A}$ computes the point-wise product of the functions which $\mathcal{A}_1$ and $\mathcal{A}_2$ compute.

Intriguingly, coordinate-wise multiplication corresponds to a similarly natural, if maybe less well-known, construction from automata theory, namely the *full Cartesian product* of two automata. Suppose we are given two automata $\mathcal{A}_i = (Q_i, \Sigma_i, I_i, \Delta_i, F_i)$, for $i \in \{1, 2\}$, then let

$$\mathcal{A} := \mathcal{A}_1 \otimes_c \mathcal{A}_2 := (Q_1 \times Q_2, \Sigma_1 \times \Sigma_2, I, \Delta, F)$$

where

$$\begin{aligned} I(q_1, q_2) &:= I_1(q_1) \cdot I_2(q_2), \\ \Delta((q_1, q_2), (\sigma_1, \sigma_2), (q_1', q_2')) &:= \Delta_1(q_1, \sigma_1, q_1') \cdot \Delta_2(q_2, \sigma_2, q_2'), \text{and} \\ F(q_1, q_2) &:= F_1(q_1) \cdot F_2(q_2). \end{aligned}$$

It is immediate to see that what $\mathcal{A}$ computes is the function f, as defined by equation (8), where $\oplus = \cdot$.

Point-wise multiplication can also be thought of as a special case of coordinate-wise multiplication in the following sense: after performing coordinate-wise multiplication, project the image to its "diagonal". That is, if the resulting automaton is input the space point $(y_1, \ldots, y_n)$, it simulates the automaton we currently have by feeding it with $(y_1, \ldots, y_n, y_1, \ldots, y_n)$.

To this end, it is sufficient to remove all transitions labeled with some symbol $(y_1, \ldots, y_n, z_1, \ldots, z_n)^{tr}$ where $(y_1, \ldots, y_n) \neq (z_1, \ldots, z_n)$. For the remaining transitions, we delete the last n components of the labeling of each, which gives us n-dimensional labelings, as desired.

6. Local operations

As the only representative of local operations, we will discuss the *convolution* of two functions. Convolutions naturally arise in image manipulation on the one hand. On the other hand, convolution is used broadly throughout mathematics as an operation in its own interest.

The convolution of two functions f_1 and f_2 (of the same dimension n), denoted by $f_1 * f_2$, is defined as follows:

$$(f_1 * f_2)(x) := \int_{[0,1)^n} f_1(w(x - t)) f_2(t) \mathrm{d}t,$$

where w is the coordinate-wise *wrap-around function*, *i.e.*,

$$w(x_1, \ldots, x_n) := (x_1 - \lfloor x_1 \rfloor, \ldots, x_n - \lfloor x_n \rfloor).$$

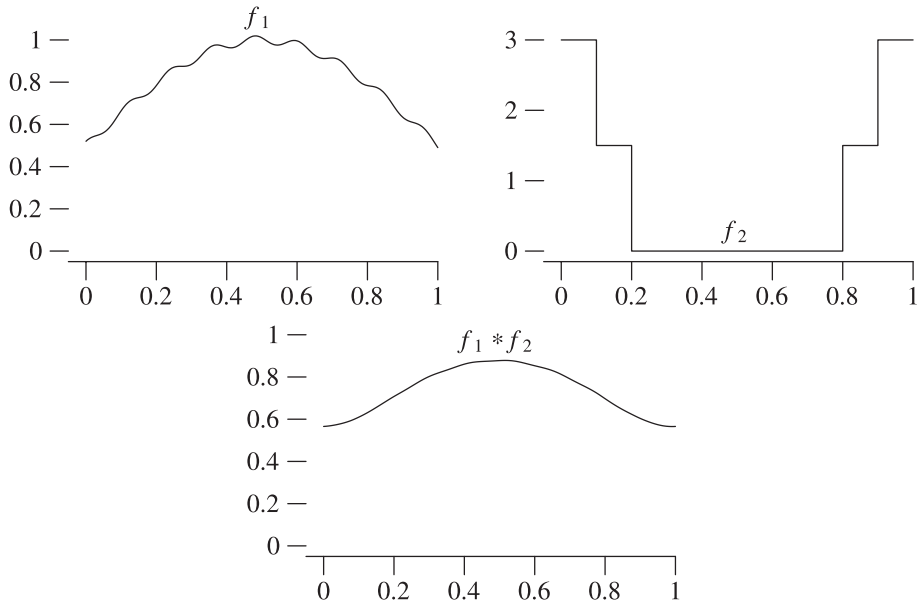


FIGURE 1. Convolution of two functions f_1, f_2.

Example 6.1. Consider the functions $f_1, f_2 \colon [0,1) \to \mathbb{R}$, given by their function graphs as in Figure 1.

The idea is that the convolution $f_1 * f_2$ should cancel the ripple of f_1 because the effect of f_2 is to average out the values of f_1 in a 0.2-ball around every x coordinate where values within a 0.1-ball receive double weight. Note that $\int_0^1 f_2(t)\mathrm{d}t = 0.9$, which means that the average value of $f_1 * f_2$ will be 10% smaller than that of f_1.

The fact that convolution, too, belongs to the class of feasible operators for automatic functions, is a corollary from these facts:

(1) For any n-dimensional automatic function f, given by the automaton computing f, we can construct an automaton computing the $(2{\cdot}n)$-dimensional function $f^w(x,t) := f(w(x-t))$.
(2) We can extend an automatic function to a higher dimension, *i.e.*, we can construct an automaton computing $\widetilde{f_2}(x,t) := f_2(t)$.
(3) Point-wise multiplication is feasible.
(4) The integral is an operation that can be carried out directly (*i.e.*, exactly and with almost negligible complexity) on an automaton.

An explanation of the first fact will be deferred to the end of this section. Fact two is an easy observation, and the third fact has been pointed out in the preceding section.

For the last fact, the reader is referred to [5]. Note, however, that the integral in [5] is defined as a unary operation on automatic functions, or, to be precise,

as the following transformation of functions:

$$\begin{aligned} \textstyle\int : f &\mapsto \textstyle\int f \\ (\textstyle\int f)(x) &:= \displaystyle\int_{\substack{0\le t_1\le x_1 \\ \vdots \\ 0\le t_n\le x_n}} f(t)\mathrm{d}t. \end{aligned}$$

In fact, [5] only deals with $n = 1$, but it is straightforward to generalize the idea to arbitrary values of n. In this notation, if we set

$$f_x(t) := f_1(w(x-t)) \cdot f_2(t)$$

for all $x \in [0,1)$, we would need to evaluate $\int f_x$ in the point $(1,\dots,1)$ in order to compute $f_1 * f_2$ in x. Formally,

$$(f_1 * f_2)(x) = (\textstyle\int f_x)(\underbrace{1,\dots,1}_{n \text{ times}}).$$

(Note that it is not significant for an integral whether it is computed over $[0,1)^n$ or over $[0,1]^n$.)

Hence, the difficulty lies in the fact that we aim at integrating a parametrized function whose parameter is the input, which – naturally – may vary. This is, however, a problem which can easily be overcome.

Note that by the average-preserving property, computing, for any finite automaton $\mathcal{A}$, the value

$$\int_{t\in[0,1)^n} \mathcal{A}(t)\mathrm{d}t = \int_{t\in[0,1]^n} \mathcal{A}(t)\mathrm{d}t = \mathcal{A}(\varepsilon)$$

is easy: $\mathcal{A}(\varepsilon)$ is exactly the average of $\mathcal{A}$ (as a function) over all of $[0,1)^n$, and this is what an integral expresses (multiplied by the measure of the set over which to integrate).

Likewise, for example, we have:

$$\int_{t=0}^{1/2} \mathcal{A}(t)\mathrm{d}t = \frac{1}{2}\mathcal{A}(\text{`0'}) \quad \text{and} \quad \int_{t=1/2}^{3/4} \mathcal{A}(t)\mathrm{d}t = \frac{1}{4}\mathcal{A}(\text{`10'})$$

where '0' and '10' denote the *words* 0 and 10, respectively (and none of the real values of zero or ten).

In fact, [5] (without mentioning it explicitly) combines this with the fact that every interval $[0,y)$ can be expressed as the (possibly infinite) disjoint union of intervals whose lengths are powers of two (with a negative exponent) and whose

boundaries have finite binary representations. Formally,

$$\big[0,y\big) = \biguplus_{\substack{k\in\mathbb{N}\setminus\{0\}\\ \mathrm{pref}_k(x^{-1}(y))\in\{0,1\}^*\{1\}}} \big[x(\mathrm{pref}_{k-1}(x^{-1}(y))), x(\mathrm{pref}_k(x^{-1}(y)))\big).$$

For example, $\frac{2}{3} = 0.1010101010\ldots_{(2)}$, and hence,

$$\left[0,\frac{2}{3}\right) = \underbrace{\left[0,\frac{1}{2}\right)}_{\text{1st digit}} \uplus \underbrace{\left[\frac{1}{2},\frac{5}{8}\right)}_{\text{3rd digit}} \uplus \underbrace{\left[\frac{5}{8},\frac{21}{32}\right)}_{\text{5th digit}} \uplus \cdots$$

It is thus possible to conclude that

$$\int_0^y \mathcal{A}(t)dt = \sum_{\substack{w\in\{0,1\}^*\\ w1\in\mathrm{Pref}(x^{-1}(y))}} \int_{x(w)}^{x(w1)} \mathcal{A}(t)dt = \sum_{\substack{w\in\{0,1\}^*\\ w1\in\mathrm{Pref}(x^{-1}(y))}} 2^{-|w|-1}\cdot\mathcal{A}(w0)$$

where $\mathrm{Pref}(z)$ is the set of all finite prefixes of some infinite word z, *i.e.*,

$$\mathrm{Pref}(z) := \{\mathrm{pref}_k(z) \mid k\in\mathbb{N}\}.$$

Note that $\mathcal{A}(w0)$ is the average function value of $\mathcal{A}$ in the interval

$$\big[x(w0),x(w1)\big) = \{y\in[0,1) \mid w0\in\mathrm{Pref}(x^{-1}(y))\}.$$

This yields the natural construction of the integral automaton (in one dimension): for every factorization $w = u1v$ of the input word w, simulate the original automaton's work on the word $u0$, but weight its output by $2^{-|u0|}$ (and add these outputs up). This can be achieved by halving the weight of every existing transition and by additionally introducing "exit transitions", labeled 1 and of a weight by which it would have been possible to read the symbol 0 (as the last symbol of a word) in the original automaton.

Formally, for $\mathcal{A} = (Q,\{0,1\},I,\Delta,F)$, set $\mathcal{A}_\int := (Q_\int,\{0,1\},I_\int,\Delta_\int,F_\int)$ where

$$Q_\int := Q\uplus\{q_\int\};$$

$$I_\int(q) := \begin{cases} I(q) & q\in Q\\ 0 & \text{otherwise;}\end{cases}$$

$$F_\int(q) := 1 \quad (\textit{cf.}\ \text{Obs. 2.8}); \text{ or equivalently,} \quad F_\int(q) := \begin{cases} 1 & q = q_\int\\ 0 & \text{otherwise;}\end{cases}$$

$$\Delta_f(q,\sigma,q') := \begin{cases} \frac{1}{2}\Delta(q,\sigma,q') & \{q,q'\} \subseteq Q \\ \frac{1}{2}\sum_{q''\in Q}\Delta(q,0,q'')F(q'') & q \in Q,\ q' = q_f, \text{ and } \sigma = 1 \\ 1 & q = q' = q_f \\ 0 & \text{otherwise.} \end{cases}$$

What we need here, however, is slightly different: we are given a finite automaton $\mathcal{A}$ that computes the (n_1+n_2)-dimensional function $g\colon [0,1)^{n_1+n_2} \to \mathbb{R}$, and we would like to construct an automaton computing the n_1-dimensional function $f\colon [0,1)^{n_1} \to \mathbb{R}$, defined by:

$$f(x) := \int_{t\in[0,1)^{n_2}} g(x,t)\mathrm{d}t.$$

For $n_1 = 0$, this task collapses to computing $\mathcal{A}(\varepsilon)$, as pointed out above. For $n_1 > 0$, it is only slightly more involved: let $\mathcal{A} = (Q, \{0,1\}^{n_1+n_2}, I, \Delta, F)$. Then, define $\mathcal{A}' := (Q, \{0,1\}^{n_1}, I, \Delta', F)$ where

$$\Delta'(q,\sigma,q') := 2^{-n_2}\sum_{\sigma'\in\{0,1\}^{n_2}} \Delta(q,(\sigma,\sigma'),q')$$

for all $q, q' \in Q$ and all $\sigma \in \{0,1\}^{n_1}$. In other words, $\mathcal{A}'$ computes the average of the function g with respect to the last n_2 components. This coincides with the function f.

The only thing left to show is that for an automaton $\mathcal{A} = (Q, \{0,1\}^n, I, \Delta, F)$ computing the function $f\colon [0,1)^n \to \mathbb{R}$, we can construct an automaton that computes $f^w\colon [0,1)^{2\cdot n} \to \mathbb{R}$. The idea is to simulate the subtraction of the two input values. For $n = 1$, imagine that the input word is

$$w = \sigma_1 \ldots \sigma_p = \binom{s_1}{s'_1} \cdots \binom{s_p}{s'_p}.$$

A written subtraction of $s'_1 \ldots s'_p$ from $s_1 \ldots s_p$ amounts to finding carry bits $b_0, \ldots, b_p$ and then computing the result bits $m_1, \ldots, m_p$ according to the definition of $m\colon \{0,1\}^4 \to \{0,1,\perp\}$ in Table 1, where b denotes the carry bit "to the left" and b' the one "to the right" of the current column.

In other words, the school method of subtracting would allow us to write a computation like this:

$$\begin{array}{ccccccccccccc} & & s_1 & & s_2 & & \cdots & & s_{p-1} & & s_p & & \\ b_0 & & s'_1 & b_1 & s'_2 & b_2 & \cdots & b_{p-2} & s'_{p-1} & b_{p-1} & s'_p & b_p & , \\ & & \multicolumn{3}{c}{\text{---------}} & & & & \multicolumn{3}{c}{\text{---------}} & & \\ & & m_1 & & m_2 & & \cdots & & m_{p-1} & & m_p & & \end{array}$$

where $b_0 = b_p = 0$. If we waive the requirement that $b_0 = 0$, we may subtract greater values from lesser ones (by means of the wrap-around function).

TABLE 1. Subtraction of bits according to known carry bits. May be undefined ($\perp$) for quadruples which cannot appear.

b	s	s'	b'	$m(b,s,s',b')$
0	0	0	0	0
0	0	0	1	$\perp$
0	0	1	0	$\perp$
0	0	1	1	$\perp$
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	$\perp$

b	s	s'	b'	$m(b,s,s',b')$
1	0	0	0	$\perp$
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	$\perp$
1	1	0	1	$\perp$
1	1	1	0	$\perp$
1	1	1	1	1

In multiple dimensions, we naturally extend m to

$$M\colon \{0,1\}^{4\cdot n} \cong \{0,1\}^n \times \{0,1\}^{2\cdot n} \times \{0,1\}^n \to \{0,1\}^n \uplus \{\perp\}$$

by setting

$$M(b_1,\dots,b_n,s_1,\dots,s_n,s'_1,\dots,s'_n,b'_1,\dots,b'_n) := \begin{cases} (m(b_1,s_1,s'_1,b'_1),\dots,m(b_n,s_n,s'_n,b'_n)) & \text{if } \perp \notin \{m(b_i,s_i,s'_i,b'_i) \mid 1 \le i \le n\} \\ \perp & \text{otherwise.} \end{cases}$$

We are now ready to give a formal definition of $\mathcal{A}^w$, the wrap-around automaton that is to compute f^w. For it to keep track of the carry bit only means to double the set of states (for every dimension). Note, however, that we are fed our input from left to right, and consequently, dealing with the carry bit can be viewed as a nondeterministic process. Set

$$\begin{aligned} \mathcal{A}^w &:= (Q \times \{0,1\}^n, \{0,1\}^{2\cdot n}, I^w, \Delta^w, F^w), \\ I^w(q,b) &:= I(q), \\ F^w(q,b) &:= \begin{cases} F(q) & \text{if } b = (0,\dots,0) \\ 0 & \text{otherwise} \end{cases}, \text{ and} \\ \Delta^w((q,b),\sigma,(q',b')) &:= \begin{cases} 0 & \text{if } M(b,\sigma,b') = \perp \\ \Delta(q,M(b,\sigma,b'),q') & \text{otherwise} \end{cases} \end{aligned}$$

for all $b, b' \in \{0,1\}^n$ and $\sigma \in \{0,1\}^{2n}$.

Note that ignoring the leftmost carry bit is realized by setting $I^w(q,b) := I(q)$ (which contrasts with the definition of F^w).

7. Conclusion

Automatic functions are a phenomenon as natural in the context of computing as are polynomial functions in the context of physics (where the notion of "being natural" is determined, *e.g.*, by asymptotic growth, derivability, etc.). Every polynomial function is automatic but not vice versa, a fact that invites further research on this intriguing class of functions. Their practical use in the context of signal processing can only underline their importance. It is a neat but rather immediate fact that the class of automatic functions is closed under operations such as the sum or the product. A more important structural observation is that it is closed under convolution. Convolutions play both an important and a natural role throughout mathematics, which suggests that the class of automatic functions deserves further investigation which would be geared toward its theoretical properties and thus extend our conception of its role even beyond its applicability in image and signal processing.

References

[1] J. Berstel and M. Morcrette, *Compact representation of patterns by finite automata*, in Proc. Pixim '89, Paris (1989) 387–402.
[2] V. Blondel, J. Theys and J. Tsitsiklis, When is a pair of matrices stable? Problem 10.2 in *Unsolved problems in Mathematical Systems and Control Theory*. Princeton Univ. Press (2004) 304–308.
[3] K. Culik II and S. Dube, Rational and affine expressions for image descriptions. *Discrete Appl. Math.* **41** (1993) 85–120.
[4] K. Culik II and I. Friš, Weighted finite transducers in image processing. *Discrete Appl. Math.* **58** (1995) 223–237.
[5] K. Culik II and J. Karhumäki, Finite automata computing real functions. *SIAM J. Comput.* **23** (1994) 789–814.
[6] K. Culik II and J. Kari, Image compression using weighted finite automata. *Comput. Graph.* **17** (1993) 305–313.
[7] K. Culik II and J. Kari, Efficient inference algorithms for weighted finite automata, in *Fractal Image Compression*, edited by Y. Fisher, Springer (1994).
[8] K. Culik II and J. Kari, Digital Images and Formal Languages, in *Handbook of Formal Languages*, Vol. III, edited by G. Rozenberg and A. Salomaa, Springer (1997) 599–616.
[9] D. Derencourt, J. Karhumäki, M. Latteux and A. Terlutte, *On computational power of weighted finite automata,* in Proc. 17th MFCS. *Lect. Notes Comput. Sci.* **629** (1992) 236–245.
[10] D. Derencourt, J. Karhumäki, M. Latteux and A. Terlutte, On continuous functions computed by finite automata. *RAIRO-Theor. Inf. Appl.* **29** (1994) 387–403.
[11] J. Karhumäki, W. Plandowski and W. Rytter, The complexity of compressing subsegments of images described by finite automata. *Discrete Appl. Math.* **125** (2003) 235–254.
[12] K. Knopp, *Infinite Sequences and Series*. Dover publications (1956).
[13] J. Kupke, *On Separating Constant from Polynomial Ambiguity of Finite Automata*, in Proc. 32nd SOFSEM. *Lect. Notes Comput. Sci.* **3831** (2006) 379–388.
[14] J. Kupke, Limiting the Ambiguity of Non-Deterministic Finite Automata. *PhD. Thesis*. Aachen University, 2002. Available online at http://www-i1.informatik.rwth-aachen.de/~joachimk/ltaondfa.ps

Communicated by J. Hromkovic.
Received September 9, 2005. Accepted May 2, 2006.