THE LOGICAL CONTEXT OF NONLINEAR FILTERING

Edward R. Dougherty
Center for Imaging Science
Rochester Institute of Technology

Robert M. Haralick
Department of Electrical Engineering
University of Washington

ABSTRACT

The mathematical structure of binary nonlinear filtering is expressed in the context of binary cellular logic and the relevance of existing image algebras is discussed. Operator properties such as antiextensivity and idempotence are examined from a discrete logical perspective, as are the classical Matheron representations. The simplicity of the operational properties is exposed by such an approach, as is the use of commonplace logic design methods for the composition and decomposition of nonlinear filters, in particular, binary morphological filters.

1. INTRODUCTION

There has been much discussion regarding the appropriate mathematical framework for image processing. While various mathematical structures have found their way into image processing, two algebraic structures have been proposed that possess the name “image algebra” (though, admittedly there have been various stages of the two algebras). One image algebra, developed by Ritter et al [1, 2], is heterogeneous, in that it contains many sorts of entities, the essential two being images and templates. The other, as developed by Dougherty and Giardina [3], possesses a heterogeneous form, and, as further developed by Dougherty [4, 5], a homogeneous form. In fact, as might be expected, the two image algebras are very similar, the key difference being the template data structure in the Ritter algebra, and, as noted in [5], the image-template operations in the Ritter algebra can be represented in the homogeneous algebra of [4] because, at bottom, a template is an array of images. The unique aspect of the Ritter algebra is a detailed study of the algebraic properties of templates. Both image algebras serve to represent image-processing operations, both linear and nonlinear. In particular, they serve as a framework for linear operations because each contains the linear algebra of matrices as a subalgebra, and they serve as a framework for nonlinear operations because each possesses the necessary lattice structure. Regarding the necessity of structure within image algebra, Dougherty and Giardina [6] take special note of the induced nature of subalgebras. But what subalgebras need to be induced? Indeed, what subalgebras should be induced given the computational nature of image processing?

In point of fact, digital images do not form a vector space relative to induced image addition and
scalar multiplication. Strictly speaking, because the gray range is discrete and finite, image addition is not even closed. Even if we ignore the finitude of the gray range, we are still confronted by discreteness, so that the whole notion of linearity, including linear operators, cannot be subsumed within any image algebra that remains faithful to digital processing. This does not mean that richer mathematical structures cannot be of use; only that one should not see these as fully exhausting the algebraic question. It should be kept in mind that digital processing involves logic gates and bounded finite 0-1 representations. If we are to stick more closely to actual processing when we propose mathematical representations, then we need to stay within the confines of digital logic, or discrete set theory.

The present study examines the logical context of nonlinear image processing. In [7] we consider the problem from both the binary and gray-scale perspectives, but here, in what is essentially the first part of [7], we stay within the binary setting. Our goal is not to present new mathematics, nor is it to present a new representation theory; rather, it is to make clear that, within the restricted framework of digital logic, the fundamental theory of nonlinear binary image filtering does not leave its logical roots. In [7], we go further and examine gray-scale filtering, showing that it, too, when practiced digitally, remains within the confines of very basic logical and algebraic structures.

Two perspectives are particularly significant to the present exposition. The first is the set-theoretic analysis of binary processing due to Matheron [8]. There, for the first time, the set-theoretic properties of binary image filtering are clearly expressed. These include monotonicity, extensivity (antiextensivity), and idempotence. These are logical (or set-theoretic) concepts and they play dominant roles in binary filtering. Matheron also carefully examines the role of translation invariance. This latter concept involves the translational structure within which image processing takes place and is closely related to image stationarity. If one reads Matheron closely, it is clear that the basic morphological operations of erosion, dilation, opening, and closing do not appear by chance in image processing. Indeed, he recognizes that any translation-invariant, monotonically increasing operator must be formed from a union of erosions and that any translation-invariant, increasing, antiextensive, and idempotent operator must be formed from a union of openings. While it is true that Serra [9, 10] and Matheron [11] have since recognized that the appropriate mathematical setting for such concepts is a complete lattice, and that in such a framework the basic Matheron propositions extend, the foundation of their work is in the binary setting (see also Heijmans and Ronse [12, 13] and -Heijmans [14]), which has direct interpretation in digital logic.

The second perspective of special import herein concerns cellular logic (see Preston et al 15,16)). Just as the central role played by mathematical morphology arises from the set-theoretic aspects of binary processing, the key role of cellular logic arises from the manner in which the binary image operators must be implemented on a digital computer. Cellular logic, and relevant related architectures, impress themselves on the algebraic analysis of image processing because the processing is digital. While it might be tempting to separate the computational and abstract mathematical problems, treating the former as architectural and the latter as algebraic, in fact they are interrelated.

The present paper begins with cellular logic and explains the manner in which algebraic binary-filter theory emerges therefrom. Such an approach naturally places those concepts typically considered to be morphological directly into the framework of cellular logic, which of course explains (in hindsight) the major role played by cellular logic in the implementation of
morphological processing. One might ask whether there is anything to be gained by such an exercise, other than some readjustment of thinking. In fact, as will become evident, there is much more to be gained. By recognizing the practical Boolean nature of morphology, we see that standard computing tools such as Karnaugh maps and Quine-McClusky reduction can be applied to the construction of morphological operators. In operator design we are confronted by both the need to compose operator representations and, on the other hand, to decompose operators into constituent parts satisfying different algebraic constraints. Having a clear appreciation of the discrete logical character of nonlinear operators allows the application of existing automatic routines.

2. CELLULAR-LOGIC FILTERS

We consider translation-invariant, moving-window operators on the space $S_B$ of binary signals defined on $Z$, the set of integers. We assume the window, $W_{<m>}$, is centered and of length $2M + 1$. If $\Psi$ is an operator of the specified type and $x = \{x[m]\}$ is a binary signal in $S_B$, then

$$\Psi(x)[m] = \Psi(x[m - M], x[m - M + 1], ..., x[m + M])$$  \hspace{1cm} (1)$$

where we do not distinguish between the operator and the function rule defining the operator, calling them both $\Psi$. Relative to computer architecture, window logic is manifested as cellular logic, and therefore the filter $\Psi$ has historically been called a cellular-logic filter. The choice of $Z$ as the domain space for $S_B$ is for convenience. Insofar as the subsequent logical analysis is concerned, the operative functional expression is equation (1) and it depends only on denumerability (discreteness) of the domain and window finiteness. In a setting different than $Z$ (say, binary images defined on $Z \times Z$), the window can be of any shape and the ordering $x[m - M], x[m - M + 1], ..., x[m + M]$ merely represents some given listing of the way in which the window is to be scanned. In particular, the assumption that the window is centered in equation (1) serves only the purpose of notational convenience.

Since $\Psi$ is translation invariant, much of its analysis can be accomplished by considering the single output value $\Psi(x)[0]$, whose value depends on the window $W_{<0>} = \{-M, -M + 1, ..., M\}$ centered at the origin. $\Psi(x)[0]$ can be considered to be a binary functional on the set $M$ of $\{0, 1\}$-valued functions defined on $W_{<0>}$. Every element of $M$ can be represented by a string of ones and zeros, $(x[-M], x[-M + 1], ..., x[M])$. From a set-theoretic perspective, every element of $M$ is a subset of $W_{<0>}$, where $j$ lies in the subset if and only if $x[j] = 1$. Union and intersection in $W_{<0>}$ are replaced by the logical maximum and minimum operations

$$x \lor y = (x[-M] \lor y[-M], ..., x[M] \lor y[M])$$  \hspace{1cm} (2)$$

$$x \land y = (x[-M] \land y[-M], ..., x[M] \land y[M])$$  \hspace{1cm} (3)$$

in $M$. Moreover, the order relation "$x \leq y$ if and only if $x_i \leq y_i$" for $i = -M, ..., M$ corresponds to the subset relation in $W_{<0>}$. As a binary functional, $\Psi(x)[0]$ can be written in logical format as a maximum of minima, or, using logical notation, as a canonical sum of products

$$\Psi(x)[0] = \sum x[-M]p[-M]x[-M + 1]p[-M + 1]...x[M]p[M]$$  \hspace{1cm} (4)$$

where, for \( j = -M, \ldots, M \), \( p[j] \) is \(-1\), 0, or 1, and where \( x[j]^{-1} \) is the negation of \( x[j] \) (also written \( x[j]' \)) and \( x[j]^0 \) means the logical variable \( x[j] \) does not appear in the factor. In other words, \( \Psi(x)[0] \) is a Boolean expression over \( 2M + 1 \) binary variables. As is well known, there are many equivalent expressions to (4), and in fact there exist methods such as Karnaugh maps and the Quine-McCluskey procedure for minimizing the number of logic gates forming canonical sum-of-product expressions.

Owing to translation invariance, the logical expression (4) applies to \( \Psi(x)[m] \) for any \( m \), the \( p[j] \) remaining the same:

\[
\Psi(x)[m] = \sum x[m - M]p[-M] \ldots x[j]p[j] \ldots x[m + M]p[M] \tag{5}
\]

The variables \( x[j] \) lie in the translated window \( W < m > = W < 0 > + m \).

Another way of looking at the expansion (4) [and therefore at the expansion (5)], is to proceed in the following manner: (1) group the variables with +1 exponents in each minterm and let \( W_1 < 0 > \) denote their product; (2) group the variables with −1 exponents in each minterm and let \( W_1 < 0 >' \) denote their product; (3) ignore all variables with 0 exponent. Then expression (4) takes the form

\[
\Psi(x)[0] = \sum W_1 < 0 > W_1 < 0 >' \tag{6}
\]

where it is possible for \( W_1 < 0 > \) or \( W_1 < 0 >' \) to be null, in which case it is denoted by 1.

Geometrically, \( W_1 < 0 > \) can be interpreted as the subwindow of \( W < 0 > \) corresponding to the positive Boolean variables (exponent +1) and \( W_1 < 0 >' \) can be interpreted as the subwindow corresponding to the negative Boolean variables (exponent −1). We will subsequently make use of this convention by considering translates \( W_i < m > \) and \( W_i < m >' \). For instance, if \( W < 0 > \) is the live-point window and

\[
\Psi(x)[0] = x[-1]x[0]x[1] + x[0]x[1] + x[-1]x[1]'x[2]' \tag{7}
\]

then \( W_1 < 0 > = x[1]x[0]x[1], \ W_1 < 0 >' = 1, \ W_2 < 0 > = x[0]x[1], \ W_2 < 0 >' = 1, \ W_3 < 0 > = x[-1], \) and \( W_3 < 0 >' = x[1]'x[2]' \). Among other things, the subwindow notation facilitates writing outputs at other points besides the origin. Here, for instance, \( \Psi(x)[m] \) is simply written as

\[
\Psi(x)[m] = W_1 < m > + W_2 < m > + W_3 < m > W_3 < 0 >' \tag{8}
\]

\( V_i < m > \) and \( W_i < m >' \) referring to the translated subwindows \( W_i < i > + m \) and \( W_i < m >' + m \), respectively.

3. INCREASING FILTERS

A cellular-logic filter \( \Psi \) is **monotonically increasing** if \( x \leq y \) implies \( \Psi(x) \leq \Psi(y) \). Owing to translation invariance, \( \Psi \) is increasing if and only if \( (x[-M], \ldots, x[M]) \leq (y[-M], \ldots, y[M]) \) implies \( \Psi(x)[0] \leq \Psi(y)[0] \). \( \Psi \) is increasing if and only if, when expressed as a sum of products, there exists no negation in the expansion; i.e., \( W_i < 0 >' = 0 \) for all \( i \). In logical
terminology, $\Psi$ is a positive Boolean function. There is a natural ordering on binary operators. Suppose $\Psi_1$ and $\Psi_2$ are two operators. We write $\Psi_1 \leq \Psi_2$ if and only if $\Psi_1(x) \leq \Psi_2(x)$ for any signal $x$. Now suppose $\Psi_1$ and $\Psi_2$ are increasing. Then $\Psi_1 \leq \Psi_2$ if and only if for any minterm of $\Psi_1$ there exists a minterm of $\Psi_2$ whose factors form a subset of the factors of the given minterm for $\Psi_1$.

An increasing cellular-logic filter $\Psi$ is said to be antiextensive [extensive] if $\Psi(x) \leq x [\Psi(x) \geq x]$ for all $x$. Relative to a sum-of-products expression for $\Psi(x)[0]$, $\Psi$ is antiextensive if and only if each minterm of $\Psi(x)[0]$ contains $x[0]$. $\Psi$ is extensive if and only if it possesses the singleton minterm $x[0]$ in its sum-of-products representation.

There exists a minimal sum-of-products expression for any operator. Hence, an increasing filter $\Psi$ has a canonical representation

$$\Psi(x)[0] = W_1<0> + W_2<0> + \ldots + W_p<0>$$

possessing a minimal number of minterms. The minimal expression is unique and can be obtained from any other sum-of-products expression simply by removing any minterm for which there is a distinct minterm whose factors form a subset of the given minterm's factors. In effect, redundant minterms are deleted from the representation. In general, any number of minterms can be adjoined to the minimal expression without changing the filter so long as each is formed from an existing minterm by adjoining factors.

4. ITERATION

Of great concern in filtering is iteration: given the filter $\Psi$, what can we say about the product $\Psi \Psi$? For the moment, we consider an arbitrary filter $\Psi$, not necessarily increasing, and we examine the sum-of-products representation for $\Psi \Psi$. The cumbersome part of the problem is this: when $\Psi$ operates on $\Psi(x)$, each variable $y[m]$ in $y = \Psi(x)$ is expressed as a sum-of-products of the original $x$ variables lying in the window $W<m>$ about $x[m]$. Thus, the expression for $\Psi \Psi(x)[0]$ potentially includes the variables $x[-2M], x[-2M + 1], \ldots, x[2M]$. The expression for $\Psi \Psi(x)[0]$ results from putting the expressions for $\Psi(x)[-M], \Psi(x)[-M + 1], \ldots, \Psi(x)[M]$ into the expression for $\Psi(x)[0]$ in place of $x[-M], x[-M + 1], \ldots, x[M]$, respectively. A key point is that once this has been done, reduction can be done to achieve a minimal-gate representation, and this can be accomplished automatically by some procedure such as the Quine-McCluskey algorithm.

If $\Psi$ happens to be increasing, the same reasoning applies; however, here reduction is much simpler. We need only expand the terms within the sum-of-products representation for $\Psi$ when we replace the variables $x[j]$ by $\Psi(x)[j]$ and eliminate redundant minterms. This can always be done automatically.

As an illustration, consider the three-point window about the origin and let

$$\Psi(x)[0] = x[-1]x[0] + x[0]x[1]$$

Then
\[ \Psi \Psi (x)[0] = (x[-2]x[-1] + x[-1]x[0])(x[-1]x[0] + x[0]x[1]) \\
+ (x[-1]x[0] + x[0]x[1])(x[0]x[1] + x[1]x[2]) \\
= \Psi (x)[0](x[-2]x[-1] + x[-1]x[0] + x[0]x[1] + x[1]x[2]) \\
= \Psi (x)[0](x[-2]x[-1] + \Psi (x)[0] + x[1]x[2]) = \Psi (x)[0] \]
the last equality following from the fact that for any logical expression \( ab \), where \( a \leq b \), \( ab = a \). For this particular example we obtain the very special relation \( \Psi \Psi = \Psi \).

5. IDEMPOTENCE

A filter \( \Psi \) is said to be idempotent if \( \Psi \Psi = \Psi \). For increasing filters, idempotence can be characterized in terms of sum-of-products expressions. Consider the minimal sum-of-products expression for an increasing filter \( \Psi \). Some minterms of \( \Psi (x)[0] \) contain \( x[0] \) and some do not. Thus, we can express \( \Psi (x)[0] \) as

\[ \Psi (x)[0] = x[0] \sum f_i(x[-M],..., x[-1], x[1],..., x[M]) \\
+ \sum g_j(x[-M],..., x[-1], x[1],..., x[M]) \tag{12} \]

where \( f_i \) and \( g_j \) are products of the variables in the centered window \( W<0> \), excluding the variable \( x[0] \). If \( x[0] \) happens to a minterm of \( \Psi (x)[0] \), then one of the \( f_i \) is 1, and without loss of generality we assume \( f_1 = 1 \). If the second sum is empty, then \( \Psi \) is antiextensive; otherwise, it is not. If \( f_1 = 1 \), then \( \Psi \) is extensive; otherwise, it is not. We write the decomposition (12) as

\[ \Psi (x)[0] = x[0]\Psi_0(x)[0] + \Psi_1(x)[0] \tag{13} \]

Operating a second time by \( \Psi \) yields

\[ \Psi \Psi (x)[0] = \Psi (x)[0] \sum f_i(\Psi (x)[-M],..., \Psi (x)[-1], \Psi (x)[1],..., \Psi (x)[M]) \\
+ \sum g_j(\Psi (x)[-M],..., \Psi (x)[-1], \Psi (x)[1],..., \Psi (x)[M]) \tag{14} \]

In terms of the decomposition (13), idempotence takes the form

\[ \Psi (x)[0] = \Psi (x)[0]\Psi_0(\Psi (x))[0] + \Psi_1(\Psi (x))[0] \tag{15} \]

which is a logical identity of the form \( a = ab + c \). A necessary condition for the identity is \( c \leq a \). Two sufficient conditions are (1) \( c = a \) and (2) \( c \leq a \).

A key subcase concerning idempotence for an increasing filter \( \Psi \) is when the operator is antiextensive. In such a situation, \( \Psi_1 \) is null, so that equation (15) is of the logical form \( a = ab \), and hence a necessary and sufficient condition for idempotence is

\[ \Psi (x)[0] \leq \Psi_0(\Psi (x))[0] \tag{16} \]
This is precisely what happened for the antiextensive filter of equation (10). For it,

\[ \Psi_0(x)[0] = x[-1] + x[1] \]  \hspace{1cm} (17)

\[ \Psi_0(\Psi(x))[0] = x[-2]x[-1] + x[-1]x[0] + x[0]x[1] + x[1]x[2] \]  \hspace{1cm} (18)

The filter of equation (10) belongs to the important subclass of all increasing, antiextensive, idempotent cellular-logic filters.

The filters in this special class are called openings, and in the context of a fixed window W<0>, will be called W<0>-openings. A W<0>-opening is defined by specifying a primitive product whose first factor is x[0]. To wit, let

\[ h_0 = x[0]x[j_1]x[j_2]...x[j_r] \]  \hspace{1cm} (19)

where 0 < j_1 < ... < j_r <= M, be the primitive product. For k = 1, 2, ..., r, let

\[ h_k = x[-j_k]x[j_1 - j_k]...x[j_r - j_k] \]  \hspace{1cm} (20)

Define the W<0>-opening \( \Psi(x) \) by

\[ \Psi(x)[0] = h_0 + h_1 + ... + h_r \]  \hspace{1cm} (21)

Since x[0] appears in every minterm, \( \Psi \) is antiextensive. Using strictly logical calculus it can be shown that every W<0>-opening is idempotent (see [7]).

Although every opening is antiextensive and idempotent, not every increasing, antiextensive, idempotent filter is an opening. For instance,

\[ \Psi(x)[0] = x[0](x[-2] + x[-1] + x[1] + x[2]) \]  \hspace{1cm} (22)

is increasing, antiextensive, and idempotent, but is not an opening.

6. MONOTONIC CELLULAR LOGIC AND BINARY MATHEMATICAL MORPHOLOGY

The advantages of implementing binary mathematical morphology in cellular-logic architectures have long been recognized; indeed, real-time morphological processing is typically dependent on cellular arrays. The success of the cellular approach is based upon the fact that binary morphological operations are actually reformulations of Boolean expressions, so that binary Minkowski (morphological) algebra is equivalent to cellular-logic algebra, which is itself simply Boolean algebra with translations. We examine this equivalence.

Suppose \( \Psi(x)[0] \) is defined by a single product,

\[ \Psi(x)[0] = x[j_1]x[j_2]...x[j_r] \]  \hspace{1cm} (23)

where -M <= j_1 < j_2 < ... < j_r <= M. Let \( A_{\Psi<0>} = \{ j_1, j_2, ..., j_r \} \) be the subset of W<0> associated with the product \( \Psi(x)[0] \). Then \( \Psi(x)[0] = 1 \) if and only if \( A_{\Psi<0>} \) is a subset of the set
corresponding to \( x \), this latter set to be denoted by \( \langle x \rangle \). In general, \( \Psi(x)[m] = 1 \) if and only if \( A_{\Psi < m} \) is a subset of \( \langle x \rangle \). Since \( A_{\Psi < m} = A_{\Psi < 0} + m \), this equivalence can be expressed in morphological terms: if we let \( \Psi^\wedge \) denote the set mapping corresponding to the logical mapping \( \Psi \), then

\[
\Psi^\wedge(\langle x \rangle) = \langle x \rangle \ominus A_{\Psi < 0}
\]  

(24)

where \( \ominus \) denotes erosion. Because the collection of 0-1 signals is isomorphic to the collection of integer subsets, \( \Psi \) and \( \Psi^\wedge \) are actually the same operator, so that equation (24) states that every single-product logical binary operator defined over the window \( W < 0 \) is equivalent to an erosion whose structuring element lies in \( W < 0 \).

More generally, a cellular-logic operator \( \Psi \) is defined by a sum of products possessing no negations if and only if \( \Psi \) is monotonically increasing. Since the logical operation \( "+" \) is equivalent to union, \( \Psi \) is a positive Boolean expression if and only if it is equivalent to a union of erosions, the structuring elements in the erosion expansion corresponding to minterms in the logical expansion. In sum, we have four equivalent conditions: (1) \( \Psi \) is a positive Boolean expression; (2) \( \Psi \) is monotonically increasing as a logical operator; (3) \( \Psi^\wedge \) is monotonically increasing as a set operator; (4) \( \Psi^\wedge \) is a union of erosions.

Define the kernel of the logical filter \( \Psi \) to be the collection \( \text{Ker}[\Psi] \) of all signals \( z \) for which \( \Psi(z)[0] = 1 \). Then \( z \in \text{Ker}[\Psi] \) if and only if there is a minterm \( x[i_1]...x[i_r] \) in the sum-of-products expansion for \( \Psi \) such that \( z[i_1] = ... = z[i_r] = 1 \), which is equivalent to saying that \( A = \{i_1,...,i_r\} \) is a subset of \( \langle x \rangle \), which in turn means that 0 lies in \( \langle x \rangle \ominus A \). Since \( A \) is one of the structuring elements forming the union of erosions comprising \( \Psi^\wedge \), 0 lies in \( \Psi^\wedge(\langle x \rangle) \). By definition, a set lies in the kernel of a set mapping if and only if the filtered version of the set contains the origin. Hence, the kernel of \( \Psi \) as a logical operator is equivalent to the kernel of \( \Psi^\wedge \) as a morphological filter.

If a set operator \( \Psi^\wedge \) is increasing and translation invariant, the Matheron representation states that \( \Psi^\wedge \) is expressed as the union of erosions by kernel elements, namely,

\[
\Psi^\wedge(S) = \cup \{S \ominus A: A \in \text{Ker}[\Psi^\wedge]\}
\]  

(25)

It was noticed by Maragos and Schafer [17, 18] and by Dougherty and Giardina [19, 20] that the kernel expression is redundant. \( \text{Bas}[\Psi^\wedge] \) is called the basis for \( \Psi^\wedge \) if (1) every element in the kernel possesses a subset in \( \text{Bas}[\Psi^\wedge] \) and (2) no two elements in \( \text{Bas}[\Psi^\wedge] \) are properly related by the subset relation. Bases are unique. If there exists a basis for \( \Psi^\wedge \), then the kernel expansion of equation (25) can be replaced by an expansion over the basis of the filter. The defining conditions of a basis mean there is no redundancy in the Matheron representation.

A monotonically increasing cellular-logic operator \( \Psi \) possesses a minimal sum-of-products representation. In that minimal form, no minterm is a proper subproduct of another minterm. But this says that no structuring element is a proper subset of another structuring element in the erosion expansion representing \( \Psi^\wedge \), which is then precisely the basis form of the Matheron representation for \( \Psi^\wedge \). Thus, in the discrete-window context, the Matheron basis representation of a translation-invariant, increasing set mapping is actually a restatement of the fact that every increasing logical operator over a finite set of variables has a minimal sum-of-products representation, the minimizing minterms being the filter basis.
In the context of the Matheron representation we see the morphological interpretation of openings. As defined by Matheron, an operator that is translation-invariant, increasing, antitensive, and idempotent is called a τ-opening. The most basic τ-opening is the elementary opening defined as erosion followed by dilation with the same structuring element: for signal \(<x>\) and structuring element A, the opening of \(<x>\) by A is defined by \(<x>\circ A = (\langle x \rangle \Theta A) \Theta A\). The morphological basis of the opening \(<x>\circ A\) consists of all translates of A that contain the origin. Consequently, if A is finite, \(A = \{0, r, \ldots, s\}\), then

\[
<x> \circ A = \cup \{<x> \Theta (A - j) : k = 0, 1, \ldots, r\} \tag{26}
\]

Letting \(h_0 = x[0]x[j_1 - j_0] \ldots x[j_r - j_0]\), we see that \(<x> \circ A\) is equivalent to \(\Psi(x)\), where \(\Psi(x)(0)\) is defined by equation (21). Hence, a cellular-logic opening (as we have defined it) is equivalent to a morphological opening.

A key advantage of the logical formulation of mathematical morphology is the ability to check properties and relationships automatically. For instance, since idempotence for binary morphological operators is equivalent to idempotence for logical operators, and since the latter characterization is machine checkable, we ipso facto have machine algorithms to check the morphological property. A second important example concerns the Matheron representation. Given the Matheron representations of several filters, the Matheron representation of an iteration can be found by the same algorithm that reduces an iteration of sum-of-product expansions to a single minimal sum of products.

### 7. CELLULAR LOGIC AND HIT-OR-MISS TRANSFORMATIONS

Positive Boolean expansions are equivalent to the Matheron representation; what about the general sum-of-products expression (4)? Let us again begin with a single product

\[
\Psi(x)(0) = x[j_1]x[j_2] \ldots x[j_r]x[i_1]'x[i_2]'' \ldots x[i_s]' \tag{27}
\]

where \(M \leq j_1 < \ldots < j_r \leq M, -M \leq i_1 < \ldots < i_s \leq M\), and there does not exist a pair of indices \(j_a\) and \(i_b\) such that \(j_a = i_b\). If we let \(A = \{j_1, \ldots, j_r\}\) and \(B = \{i_1, \ldots, i_s\}\), then \(\Psi(x)(m) = 1\) if and only if \(A + m\) is a subset of \(<x>\) and \(B + m\) is a subset of \(<x>^c\), the complement of \(<x>\). But this means that \(m\) lies in both \(<x> \Theta A\) and \(<x>^c \Theta B\). But the intersection of these two erosions is the hit-or-miss transform (Serra [21]) generated by the structuring pair \((A, B)\):

\[
(<x> \Theta A) \cap (<x>^c \Theta B) \tag{28}
\]

and \(\Psi\) is equivalent to the hit-or-miss operator.

If we now consider the most general form of the sum-of-products Boolean expression in equation (4), we see that every translation-invariant, moving-window binary logical function is equivalent to a union of hit-or-miss operators with structuring elements in the window. Thus, a general Boolean operator \(\Psi\) possesses a morphological equivalent \(\Psi^A\). In the discrete, moving-window case, minimal-gate expressions can be found by considering the operator mapping as a sum of products and applying some reduction algorithm.

As an illustration of how to employ the logic-morphology isomorphism, consider a four-point
image window with the origin in the lower left corner, so that \( W^{0} = \{(0, 0), (0, 1), (1, 0), (1, 1)\} \) in the Cartesian grid. If we let \( x, y, z, \) and \( w \) denote the left-right, top-down raster scan of the four-point square, then every moving-window operator can be defined by a truth table consisting of strings of the form xyzw, where the operator \( \Psi \) takes the form \( xyzw \rightarrow \Psi(\text{xyzw}) \). Suppose we wish find the minimal morphological implementation of \( \Psi \), where \( \Psi \) is defined by the truth table output (in the usual order): 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1. The table corresponds to the Karnaugh map of Figure 1, so that we have the reduced expression \( \Psi(\text{xyzw}) = xy' + yz \). Its morphological equivalent is given by

\[
\Psi^\wedge(S) = [(S \ominus A) \land (S \ominus B)] \lor [S \ominus C] \tag{29}
\]

where \( A = \{(1, 1)\} \), \( B = \{(1, 1)\} \), and \( C = \{(0, 0), (1, 1)\} \).

8. CONCLUSION

Finite-window, binary nonlinear image filtering, in particular, mathematical morphology, is fully characterized in the context of cellular logic, and the fundamental nonlinear-filter properties, including representations, are described by classical logical formulations. Aside from architectural considerations, which have historically treated the matter in this way, there are algorithm-design advantages because the tools of logic design fully apply. This can be particularly important for filter composition and decomposition, especially when iterations must possess component parts with certain algebraic properties.

9. REFERENCES


