TECHNICAL REPORT NO. 296

Lukasiewicz Logic Arrays

by

Jonathan W. Mills, M. Gordon Beavers and Charles A. Daffinger

Revised: March 1990

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY

Bloomington, Indiana 47405-4101

Łukasiewicz Logic Arrays

Jonathan Wayne Mills
M. Gordon Beavers
Charles A. Daffinger

Indiana University
Bloomington, Indiana 47405

Abstract

Łukasiewicz logic arrays (ŁLAs) are massively parallel analog computers organized as binary trees of identical processing elements performing either implication (→), negated implication (↛) or both. We have designed and built a working 31-cell CMOS VLSI ŁLA whose cells perform implication (→). In this paper we discuss the representation completeness of Łukasiewicz logic with respect to other multiple-valued logics, describe the architecture of the prototype ŁLA, its relationship to cellular automata and its VLSI implementation, show how the prototype ŁLA is programmed, and report on results obtained by programming the prototype ŁLA as a fuzzy function generator. Because ŁLAs have both an algebraic and a logical operational semantics, they can be used to implément approximate reasoning systems, including expert systems and neural networks.

1. INTRODUCTION

Łukasiewicz logic arrays (ŁLAs) are massively parallel analog computers. They are organized as binary trees of identical processing elements (called PEs, or cells), each PE performing either Łukasiewicz implication (→), negated implication (↛) or both.

We have designed and built a working 31-cell CMOS VLSI ŁLA whose cells perform implication (→). In this paper we discuss the representation completeness of Łukasiewicz logic with respect to other multiple-valued logics, describe the architecture of the prototype ŁLA, its relationship to cellular automata and its VLSI implementation, show how the prototype ŁLA is programmed, and report on results obtained by programming the prototype ŁLA as a fuzzy function generator.

The success of the prototype has encouraged us to continue research in the design and application of ŁLAs. During this research we have observed that ŁLAs offer advantages as massively parallel analog computers.

1.1 ADVANTAGES

ŁLAs are regular VLSI architectures. The VLSI implementation of ŁLAs is simple and area-efficient because they are derived from cellular automata and implemented with analog rather than digital processing elements. Although ŁLAs are analog computers they can be made surprisingly precise (5 to 8 bits), due to the simplicity of their processing elements and the accuracy of VLSI process technology.

ŁLAs are inductive architectures, which means that they can be expanded by adding more processing elements without redesigning the interconnection network. While small ŁLAs can be used as circuit components, large ŁLAs can be used as massively parallel computers. Larger ŁLAs can be created by cascading individual ŁLAs.

The general-purpose nature of ŁLAs is theoretically well-founded. Multiple-valued logics used in computational networks are capable of both symbolic and algebraic computation. ŁLAs can implement fuzzy inference and expert systems [1], neural networks [2, 3], and algebraic functions [4, 5]. Viewed as circuit components, ŁLAs are the multiple-valued logic equivalent of programmable logic arrays (PLAs) for Boolean logic.

1.2 DISADVANTAGES

Of course, ŁLAs are not ideal analog processors, but we are working to reduce their drawbacks.

The prototype ŁLAs are programmed using normal forms of sentences in the Łukasiewicz logic. This introduces data inputs on the order of $O(2^n)$ for sentences in n implications, and limits the size of the sentences that can be evaluated by a given ŁLA. Using the normal form also increases the number of pins needed on the VLSI package, far beyond the number available even in the foreseeable future. However, many data inputs are *true* or *false*, or are composed of a repeated number of variable inputs. Based on this observation we are designing ŁLAs that have external control inputs, and a restricted number of external data inputs. The data inputs are replicated and selected internally at each input of the processor array according to the externally applied control inputs.

Because ŁLAs are a new form of computational engine their use is still being studied. We have only a basic understanding of the programming methodology for ŁLAs. For example, the theoretical applicability of ŁLAs as neural networks does not immediately lead to the construction of algorithms for back-propagation.

ŁLA programming is an instance of the more general problem of programming analog and hybrid digital-analog computer architectures. Research in this area stopped about 1970 due to the dominance of digital computers. Because ŁLA-based systems will be either analog or hybrid digital-analog computers we must develop programming languages for them. Mills and Faustini [6] have proposed a language for ŁLA-based systems, but its operational semantics are still only partly defined. Completion of the semantics will require a more exact characterization of the dynamic behavior of ŁLAs, particularly ŁLAs with cyclic interconnections.

The next section describes Łukasiewicz logic and its representation completeness relative to the class of multiple-valued logics whose valuation functions can be defined in terms of +, −, *min* and *max*.

2. THE MULTIPLE-VALUED LOGICS CLASSES $L_{[0,1]}$ AND $L_{\{+,-,\wedge,\vee\}}$

(Lukasiewicz and Tarski 1930) contains a compendium of the results of investigation into multiple-valued logics obtained by Lukasiewicz and his students in the 1920's. Following the initial efforts of Lukasiewicz and Post other multiple-valued logics were developed, both discrete and continuous. Summaries of these logics can be found in (Rescher 1969) and (Gaines 1976). Most of these logics belong to $L_{[0,1]}$ which is given by:

Definition 1. *A logic L is a member of the class $L_{[0,1]}$ iff there is a logical matrix M appropriate for L with $\mathrm{M} = <\mathrm{P}, D>$ where P is a non-empty algebra whose carrier set is a subset of the real number range [0,1], with D, the set of designated elements, a non-empty proper subset of the carrier set.*

The class $L_{[0,1]}$ can be further restricted to yield a class of logics whose valuation functions for the connectives can be expressed in terms of addition, subtraction, maximum, and minimum alone. This class will be denoted by $L_{\{+,-,\wedge,\vee\}}$.

Definition 2. *A logic L in $L_{[0,1]}$ is in $L_{\{+,-,\wedge,\vee\}}$ iff all sentences of L can be evaluated using only the operators $+, -, max, min$ on the values of the atomic sentences of L.*

To appear in the Proceedings of the IEEE 20th International Symposium on Multiple-Valued Logic

The importance of the class $L_{\{+,-,\wedge,\vee\}}$ is that it contains only those logics whose sentences can be easily evaluated by analog circuits using electrical current to represent the values of logical variables. The valuation functions of these logics can be implemented by adding and subtracting electrical currents – simple operations for electronic devices: and by utilizing Ohm's law, Kirchoff's law and the law of conservation of energy to implicitly implement the operations *max* and *min* for electrical currents using the physical properties of the circuit. Furthermore, this class contains logics, such as fuzzy logic, whose significance to the fields of approximate reasoning and artificial intelligence is well established.

2.1 REPRESENTATION COMPLETENESS OF LUKASIEWICZ LOGIC RELATIVE TO $L_{\{+,-,\wedge,\vee\}}$

That Łukasiewicz logics are members of the class $L_{\{+,-,\wedge,\vee\}}$ follows from:

Definition 3. *Ł is a Łukasiewicz logic if Ł has a model* $M =< A, D >$ *where* $A =< S, \neg, \rightarrow >$ *and* S *is a subset of* $[0, 1]$ *such that:*

1. $1 \in S$,
2. *If* $x, y \in S$ *then* $min(1, 1 - x + y) \in S$, *and* $1 - x \in S$,

where $x \rightarrow y$ *and* $\neg x$ *are evaluated as* $1 - x + y$ *and* $1 - x$ *respectively.*

If $S = [0, 1]$ we get the classical propositional calculus. If S has n elements then we get the n-valued Łukasiewicz propositional calculus $Ł_n$. If the cardinality of S is $\aleph_0$ or $\aleph_1$ we get $Ł_{\aleph_0}$ or $Ł_{\aleph_1}$ which happen to have the same set of theorems. We designate $Ł_{\aleph_0}$ by Ł and take S for Ł to be the set of rational numbers between 0 and 1.
(Giles 1976) shows that Zadeh's seminal work on fuzzy set theory is closely related to Łukasiewicz logic. That Ł is representation complete with respect to the class of logics $L_{\{+,-,\wedge,\vee\}}$ follows from the fact that the evaluation of formulae in Ł has the following properties:

1. $v(\neg(\phi)) = v(\phi \rightarrow 0)$.
2. $max(v(\phi), v(\psi)) = v((\phi \rightarrow \psi) \rightarrow \psi)$.
3. $min(v(\phi), v(\psi)) = v(\neg((\neg\phi \rightarrow \neg\psi) \rightarrow \neg\psi))$
4. $min(1, v(\phi) + v(\psi)) = v(\neg\phi \rightarrow \psi)$

With the ability to perform these calculations, we have the first step in our justification of the LLA as a fundamental circuit for approximate reasoning.

2.2 McNAUGHTON'S THEOREM

McNaughton's theorem allows us to use Ł at different levels of abstraction, in particular as a classifier for elements of fuzzy sets. By showing that valuation functions for connectives in sentences in Ł are equivalent piecewise to polynomials of degree one that map the hyperspace $[0, 1]^n$ into the interval $[0, 1]$, the capability of building fuzzy pattern recognizers is provided. Thus a series of sentences in Ł defines the polytope of some simply connected solid in hyperspace of degree n. This allows us to express arbitrarily complex membership relations in logical form; in a VLSI circuit we define the polytope with a sentence from Ł, which is converted to the normal form (derived in section 4) of the sentence representable by one or more LLA circuits. The LLA circuit may be "programmed" to deal with variants of the original sentence by assigning incoming data to specific circuit inputs.

Theorem 1. (McNaughton 1951) *Let* $u_1, ..., u_n$ *be numerical variables and* $x_1, ..., x_n$ *be propositional variables. For a function* $f(u_1, ..., u_n)$ *there is a logical formula* ϕ *of* Ł *such that* $f(u_1, ..., u_n) = v(\phi(x_1, ..., x_n))$ *iff*

(i) f *is continuous over* $[0, 1]^n$ *and* $Range(f) \subseteq [0, 1]$, *and*
(ii) *there is a finite number* t *of distinct polynomials* $\pi_1, ..., \pi_t$ *each of the form*

$$\pi_j = b_j + m_{1,j}u_1 + \cdots + m_{n,j}u_n$$

with b_j, $m_{i,j}$ *integers such that for every* $< u_1, ..., u_n >$ *there is a* j *such that*

$$f(u_1, ..., u_n) = \pi_j(u_1, ..., u_n).$$

Next, homogeneous, heterogeneous and logical cellular automata are defined, and on this basis a Łukasiewicz logic array is developed that implements implication for Ł.

3. ARCHITECTURE

3.1 DESIGN

Łukasiewicz logic arrays resulted from research into cellular automata as parallel architectures for logic programming.[1] Cellular automata are of particular interest because they lead to area-efficient VLSI architectures. Such architectures are implemented as regular arrays of processing elements which communicate the results of their computation locally. They are derived by instantiating a portion of a cellular automaton as a VLSI circuit. The structure and function of the circuit arises from the definition of a cellular automaton:

Definition 4. [7] A cellular automaton C is defined by the quadruple (S, g, h^t, ϕ) where:

- S — *is a two-dimensional cellular space defined by the set of cells* $\alpha \in I \times I$ *where I denotes the set of integers.*
- g — *is a neighborhood function mapping* $S \rightarrow 2^S$ *such that* $g(\alpha)$ *is a set defining the cells in the neighborhood of* α *relative to* α. *Typically* α *is a member of its own neighborhood.*
- h^t — *is a neighborhood state function at some time t. Values of cells in the neighborhood of* α *at time t are obtained by applying* h^t *to the neighborhood* $g(\alpha)$. *The successive states of* α *at times* $\{t_0, t_1, t_2, ...\}$ *can be defined by the composition* $f \circ h^t \circ g(\alpha) = v^{t+1}(\alpha)$.
- ϕ — *is a finite automaton replicated in each cell of S and defined by the triple (V, v_0, f). V is the set of states possible for each cell,* v_0 *a distinguished quiescent state, and f a transition function mapping n-tuples of elements of V into V. The transition function f is constrained to preserve quiescence locally by requiring* $f(v_0, v_0, ..., v_0) = v_0$.

A cellular automaton is *homogeneous* if the neighborhood function and the finite automaton are identical for all cells in the cellular space S at all times *t*, otherwise the cellular automaton is *heterogeneous*.

Heterogeneous cellular automata model a wide variety of parallel computational devices. Examples include the systolic architectures of Kung and Lieserson [8], the stochastic neural machines of Alspector et. al. [9, 10] and the analog VLSI computers of Mead [11].

Ideal Łukasiewicz logic arrays (ŁLAs) are heterogeneous cellular automata that implement a denumerably infinite sentence schema of Ł. The sentence schema of Ł and the cellular automaton C are related by requiring the logical variables of Ł to correspond to cells in the cellular space S, the structure of the sentence schema to correspond to the neighborhood function g, and the connectives of Ł to correspond to the transition function f of ϕ. Ł is therefore a logic in the sense used by Belnap — an *organon*, or a tool for inference — and not a formal axiomatic theory [12].

Real Łukasiewicz logic arrays are derived by restricting the denumerably infinite sentence schema of Ł to a finite sentence schema, and implementing the finite cellular automaton that results as a direct correspondence architecture. The structure of the resulting ŁLA is dependent on its interconnection network. The prototype ŁLA uses an H-tree network whose nodes are the processing elements corresponding to the connectives in the finite sentence schema. The H-tree network was selected for its efficient use of area on a VLSI circuit, as first proposed by Leiserson [13].

1 In the general sense of "programming with logic" rather than the restricted sense of implementing Prolog.

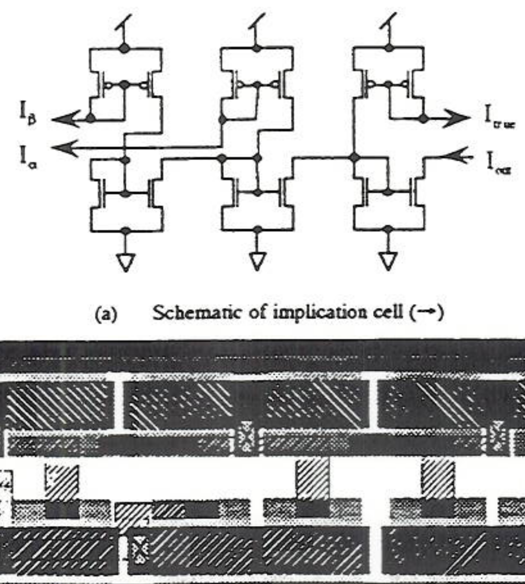
3.2 VLSI IMPLEMENTATION

Łukasiewicz logic arrays are implemented with analog processing elements. A cell in the ŁLA is implemented as an analog current-mode device performing addition, subtraction, *min* and *max* on currents. Addition and subtraction are done instantaneously, though the circuit needs a short time to stabilize.[2] Early in our work we learned of a series of fuzzy functions implemented by a basic logic cell [1]. The circuits which implement these functions also implement the algebraic valuation functions for Ł. For our purposes the most useful of Yamakawa's circuits are implication (→), which computes $min(1, 1 - \alpha + \beta)$, and bounded difference, which computes $max(0, \alpha - \beta)$. Algebraically reducing the expression for negated implication ($\neg(\alpha \rightarrow \beta)$, or $\alpha \nrightarrow \beta$) from $(1 - min(1, 1 - \alpha + \beta))$ yields $max(0, \alpha - \beta)$, showing that it is equivalent to the bounded difference.

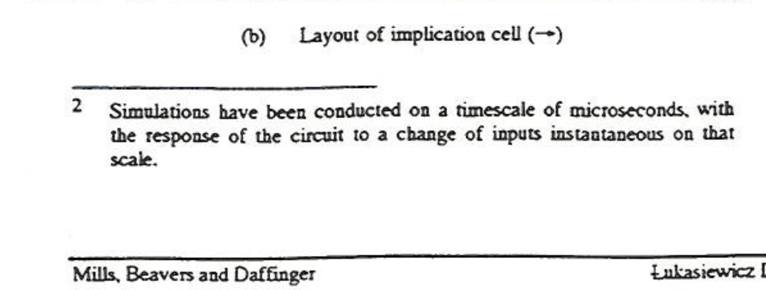
The design uses Kirchhoff's laws to sum currents at points within the ŁLA cell. To ensure that the varying current drains of adjacent cells do not affect the computation of their predecessors, as well as guaranteeing a proper input voltage, each cell is isolated by a set of current mirrors. MOS FET current mirrors have very good accuracy in making any number of copies of a given input current without placing a variable drain on that input current.

The basic cell consists of six current mirrors, and performs Łukasiewicz implication (→). A cell has two inputs and a single output, and is designed to be tiled in an H-tree (Figure 1a). The basic cell uses 11 transistors, and is 35μ by 114μ using the 2μ SCPE technology provided by the MOSIS fabrication service (Figure 1b). Basic cells are combined in an H-tree to form the ŁLA (Figure 1c).

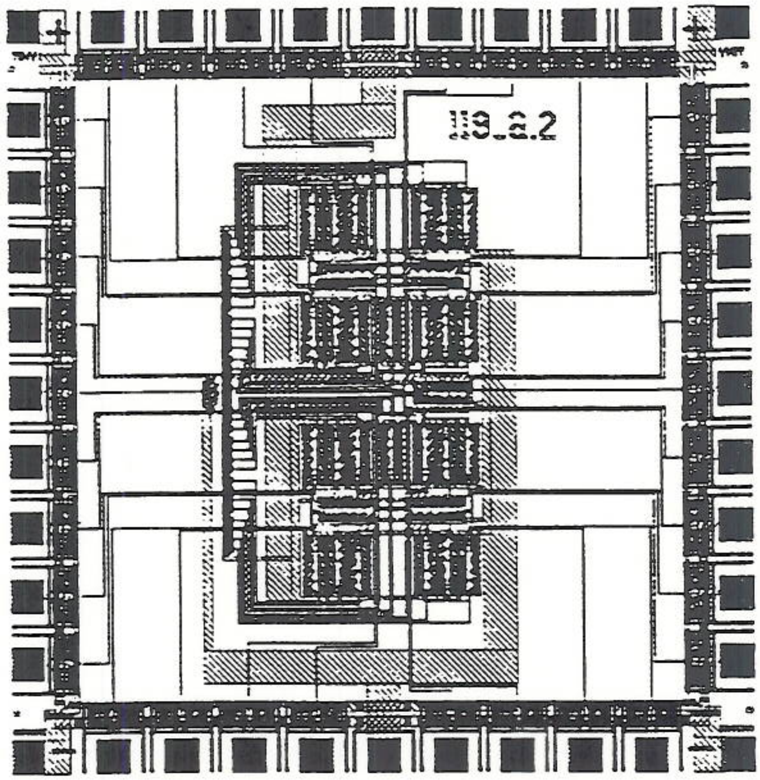
The operating range for the ŁLA cell varies from 0 to 7 volts with input and output currents varying from 0 to 20 microamperes (μA). Within this range the accuracy of the ŁLA is affected by three sources of error. The first is steady-state error, which is dependent on the actual dimensions of the transistors and other process parameters for a particular MOSIS run. The second source of error is temperature dependent, and varies as the temperature changes over long periods of time. As long as the temperature of the system in which the ŁLA is placed varies uniformly this error can be ignored. The third source of error is transient error which arises when large current swings occur in the inputs of the ŁLA and lasts until the cell has stabilized.



(a) Schematic of implication cell (→)



(b) Layout of implication cell (→)



(c) Layout of a 31-cell 5-level ŁLA

Figure 1. Heterogeneous ŁLA in implication (→)

SPICE simulations indicated that steady-state error is well-behaved, and remains within 1.5% mean and 4% maximum for small cells, growing slowly as the depth of the ŁLA is increased. Our observations agree with the simulation.

The transient error is dependent upon how chaotic the inputs of the circuit are. This is related to the number of inputs that change during a sampling interval, the amount by which they change, and the level of current used for the *true*, or maximum value. Selecting a high value for *true* increases the precision of the ŁLA but at a price: larger current swings will require a longer settling time, and produce a slower circuit.

Choosing an analog processing element yields several advantages. Because the ŁLA is a current-mode circuit it has a precision which is not achievable with an equivalently-sized voltage-mode circuit. Although Ł is infinitely valued, in practice only $Ł_2$ through $Ł_{256}$ can be implemented due to device error and the resolution of our measuring devices. The output error measured for the prototype ŁLAs is in the range of 0.25% to 2%. This gives an information density ranging from 5.6 to 8.6 bits, or approximately 50 to 400 discreet values per ŁLA. This is a useful precision for approximate reasoning systems.

The processing elements are simple, performing only Łukasiewicz implication (→) to evaluate the sentences in Ł defined by the schema. Processing elements need only two input wires and one output wire because they use analog values. Thus, the bus structure of the ŁLA is also area-efficient.

The total area used by an ŁLA is much less than the area required for an equivalent digital processor. This is based on the number of transistors needed to implement the digital processor's arithmetic logic unit (ALU) and register file, but not its control and bus interface circuitry. If each processing element has eight bits of precision, then the LIBRA digital ALU [14, 15] uses 935 times more transistors and is 1,020 times larger than the basic cell of the ŁLA (Table 1).

2 Simulations have been conducted on a timescale of microseconds, with the response of the circuit to a change of inputs instantaneous on that scale.

Table 1. Comparison of the Łukasiewicz logic array to a digital Prolog processor

	ŁLA	LIBRA	Increase
Transistors	11	10,288	935×
Area	$6.272\mu^2$	$6.4 \times 10^6\mu^2$	1,020×

However, one drawback to an area-efficient circuit is that it is limited by the number of pins available on existing VLSI circuit packages. Although an array of 1024 Łukasiewicz implication (→) cells could easily fit onto a 4500μ × 2300μ chip, it would require 2048 input pins and 1 output pin. This is 1,921 more pins than are available on a 128 pin-grid array package. Our research has shown that many functions implemented with ŁLAs will have more than half of their inputs tied to *true* or *false*. For these functions Ł)LAs can be built that use a programmable interconnection network to route internally replicated *true* and *false* inputs to the PE array. Data inputs also tend to be used more than once, so they could be internally replicated and routed, too. This approach allows large ŁLAs to fit into existing VLSI packages.

The ŁLAs described here resemble 1960's-era analog and hybrid digital-analog computers. This leads to the view of ŁLA programming as an instance of the more general problem of programming analog and hybrid computer architectures. We develop a low-level ŁLA programming methodology in the following section.

4. PROGRAMMING IN ŁUKASIEWICZ LOGIC

ŁLAs are programmed at the lowest level by fixing an interconnection network for the inputs, and presenting inputs that are either *true*, *false*, or variable. Because it is not practical to build an ŁLA for each sentence in Ł, it is necessary to develop a normal form that maps arbitrary sentences onto some general ŁLA.

4.1 A BALANCED NORMAL FORM FOR Ł

The prototype ŁLA is structured as a binary tree whose nodes are connectives, and whose leaves are logical variables. Most sentences in Ł do not map directly to this schema, but must be transformed to equivalent sentences which do. This general form of a sentence in Ł is the balanced normal form in implication, with explicit negation possible anywhere in the sentence.

Definition 5. *A sentence in Ł is in balanced normal form in implication if there exists some designated implication in the sentence, starting at which a binary tree of implications can be extracted, and for which at each non-leaf node in the tree the number of implications and logical variables in each subtree rooted at that node is equal.*

Theorem 2. *Any sentence in Ł can be rewritten to an equivalent sentence in balanced normal form in implication.*

Production of this balanced normal form can be viewed as an inverse operation of the minimization of Allen and Givone [16]. The circuit implements balanced normal form sentences in Ł because it is structured as an H-tree. The use of a binary tree to realize n-input R-valued functions for multiple-valued logic circuits was described by [17].

4.2 NEGATION-FREE NORMAL FORM

The next step toward developing a useful normal form is the transfer of negation from an arbitrary point in any sentence, moving it to either the root or one or more leaves in the binary tree of connectives and logical variables.

Unfortunately, we suspect that no normal form for Łukasiewicz logic exists with negation moved to either leaf or root implications. But it is just as useful from a computer architect's point of view to leave the negation in place as long as the negated expression $\neg\alpha$ can be re-written to an equivalent form that does not use negation explicitly, namely $\alpha \rightarrow$ *false*. A clause expressed in only one connective, while textually more complex, may be mapped to smaller and simpler physical devices that perform negation using data inputs alone.

The balanced negation-free normal form is obtained by removing negation from any sentence of Ł by simplification where possible, or by rewriting negation as $\alpha \rightarrow$ *false* otherwise. To define the balanced negation-free normal form we first define a *negation-free normal form* as follows.

Definition 6. *A sentence in Ł is in negation-free normal form iff it is expressed only in implication, and contains some designated connective such that a binary tree of connectives can be constructed whose root is the designated connective and whose leaves are logical variables in the clause.*

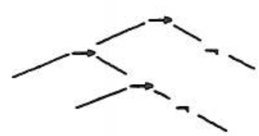
To continue the transformation the concept of the *weight* of a tree must be defined (it was implicit in Definition 5 of the balanced normal form). From this it is a short step to the definition of the balanced negation-free normal form *(BNF normal form)* and an equivalence theorem.

Definition 7. *The weight of a tree is the number of connectives and logical variables contained in the tree.*

Definition 8. *A clause is in BNF normal form iff it is in negation-free normal form, and at each non-leaf node in the tree the weights of each subtree are equal.*

Theorem 3. *Any sentence in Ł can be transformed to BNF normal form.*

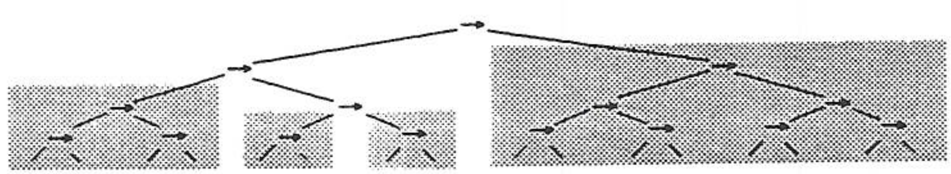
The proofs of Theorems 2 and 3 are omitted, but an example provides their substance. Consider the transformation of an arbitrary sentence in Ł to BNF normal form. The sentence is unbalanced initially, and contains negation (Figure 2).



$$(\alpha \rightarrow (\beta \rightarrow \neg\gamma)) \rightarrow \neg\delta$$

Figure 2. Unbalanced sentence in Ł

The resultant BNF normal form to which it is transformed is shown next (Figure 3). Although the textual form of the sentence is more complex, the BNF normal form uses cells of the ŁLA that the first form would have left unused. These "extra" inputs and implications can be used to adjust the constraints under which the sentence is true.



$$(((T \rightarrow T) \rightarrow (T \rightarrow \alpha)) \rightarrow ((T \rightarrow \beta) \rightarrow (\gamma \rightarrow F))) \rightarrow (((T \rightarrow T) \rightarrow (T \rightarrow \delta)) \rightarrow ((T \rightarrow T) \rightarrow (T \rightarrow F)))$$

Figure 3. BNF normal form of sentence in Figure 2

4.3 COMPLEXITY OF THE NORMAL FORM

The proof that the number of logical variables and connectives in the normal form is of complexity $O(2^n)$ where n is the minimum height of the trees formed from an arbitrary sentence of Ł is outlined: simplify all negations, then treat any negation remaining as a node in the tree; generate a set of trees by successively designating each implication in the sentence as the root connective, then select *n* equal to the minimum of the height of all generated trees. The number of inputs is at most 2^{n+1}, and the number of nodes in the tree is 2^n. Although the presence of exponential complexity in both normal forms is disturbing, some optimizations are possible. For example, if a sentence in Ł is transformed to BNF normal form, many of the inputs on the original degenerate branch are either *true* or *false*. When a normal form is so large that it spans multiple VLSI circuits, then it is possible to remove the *true* and *false* inputs by supplying the single value instead of computing it with a series of ŁLAs.

5. APPLICATIONS OF ŁLAS

Łukasiewicz logic arrays were first proposed to evaluate sentences in Ł, but because Łukasiewicz logic describes other forms of approximate reasoning, Ł)LAs are useful for a variety of applications. The dual logical and algebraic semantics of Ł allow ŁLAs to implement expert systems, neural networks [10, 18], and fuzzy computers [19, 20]. We present schematic examples for each application, and report the results obtained by programming the prototype ŁLA as a fuzzy function generator.

5.1 EXPERT SYSTEMS

ŁLAs implement expert systems by mapping membership functions to processing elements at lower levels in the array, and rules to processing elements higher in the array. A rule is a single tree that is true or false to a degree that depends on its inputs. Rules can be designed that do not fire unless their inputs reach a desired confidence level (Figure 4).

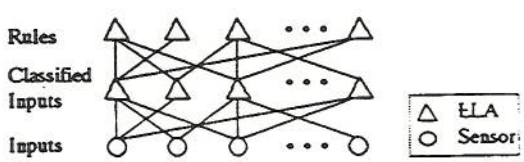


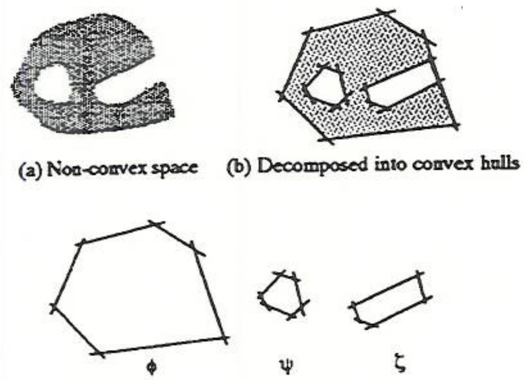
Figure 4. ŁLA implementation of expert system

A single Łukasiewicz logic array can implement a simple expert system, which may be used to embed limited "intelligence" in individual sensors. Within the array some processing elements operate at a low level of abstraction, evaluating membership in a fuzzy set, while other processing elements operate at a higher level of abstraction, implementing a rule for that sensor. The rule's operation may vary based on control inputs to the ŁLA. The expert system evaluates its sensor's input, firing the rule if the confidence factor is exceeded.

5.2 NEURAL NETWORKS

McNaughton's theorem (see Section 2) and Giles' Logic of Assertions [21] relate sentences in Łukasiewicz logic to piecewise-linear functions and the theory of convex analysis. This is the functional domain of pattern recognizers and classifiers (Figure 5), which has encouraged us to investigate neural networks implemented with ŁLAs.

Early models of nerve nets were described by McCulloch and Pitts [22]. Kleene and von Neumann anticipated much of the present-day work in neural networks, offering theoretical descriptions of the events representable in neural networks [23], and the creation of reliable computing systems from unreliable components [24].



(c) Expressed as a sentence in Łukasiewicz logic:

$$\phi \wedge \neg\psi \wedge \neg\zeta$$

Figure 5. Relationship of non-convex space to Łukasiewicz logic

The evaluation formula for Łukasiewicz implication shows how it may be used to construct a very simple "neuron." In the expression $min(1, 1 - \alpha + \beta)$, α is an inhibitory input that lowers the "firing rate", or truth value. β is an excitatory input that increases the "firing rate." Recursively connecting several implication cells produces a "neuron" with a variable threshold (Figure 6a). Summation units can also be devised (Figure 6b).

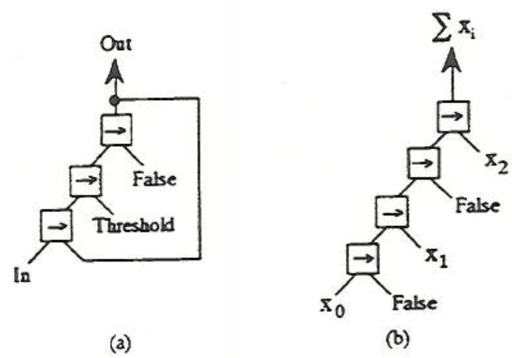


Figure 6. Implementing a "neuron" and summing element with an ŁLA

Simulations of interconnected ŁLA "neurons" and summing elements show that they have the basic properties needed to construct a neural network. The behavior of a "neuron" can be changed by modifying its threshold. For example, the slight delay before the second output pulse in the simulation is due to an intermediate "neuron" (Figure 7).

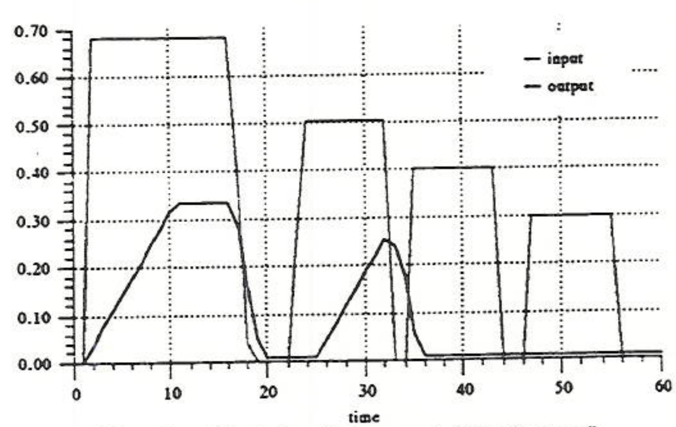


Figure 7. Simulation of interconnected ŁLA "neurons"

We are now working to construct a trainable neural network from these basic components. The initial version will be a hybrid digital-analog system, similar to those of Alspector and Graf [9, 10, 25]. However, we hope to devise analog-only systems using double-poly capacitors as storage elements for weights.

5.3 FUZZY LOGIC

Łukasiewicz logic is closely related to fuzzy logic [19, 21, 26, 27, 28]. Yamakawa shows designs for fuzzy inference engines and expert systems which may be embedded in Łukasiewicz logic arrays [1, 20, 29]. We have used the prototype ŁLA to compute fuzzy membership functions, and present the results obtained along with observations on the error measured using the prototype ŁLA.

An interesting fuzzy membership function is the "notch", defined by the expression $(\neg \alpha \rightarrow \alpha) \rightarrow \neg (\alpha \rightarrow \neg \alpha)$. This membership function was programmed into the 31-cell ŁLA as the following 32-element vector:

$$(T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,\alpha,F,T,\alpha,T,\alpha,\alpha,F,T,T,T,F).$$

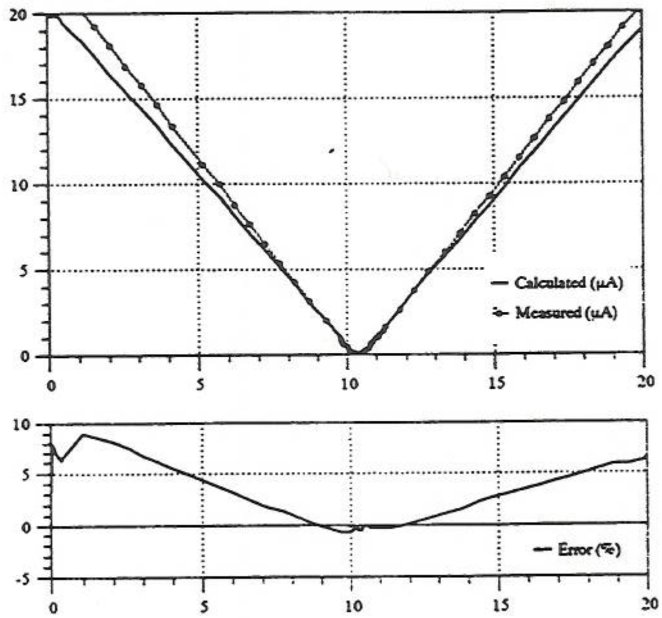
The output of the ŁLA was measured over the operating range of 0 to 20 µAmperes (µA) by varying α. The membership function was also calculated after adjusting the evaluation function for Łukasiewicz implication to the operating range of the circuit (Figure 8a).

The result of this experiment showed that the ŁLA implemented the notch function linearly, but with a slope that varied from that of the calculated function. This scaling is due to an arbitrary choice of a resistor in the measuring circuit. By changing this resistance ("trimming"), the output can be adjusted to produce a much closer fit. We have calculated that an optimally trimmed ŁLA would have a typical error of less than 2%, and a mean error less than 0.5% (Figure 8b).

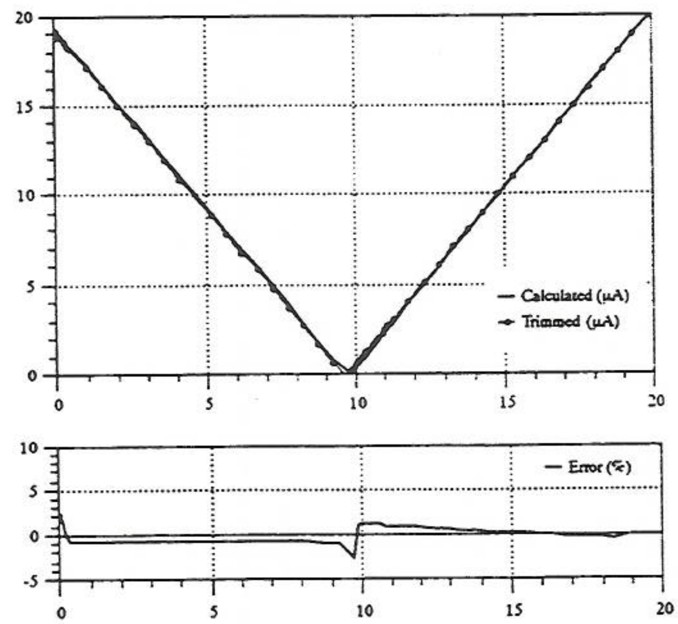
Several factors contribute to the observed error. The *true* reference for implication (→) only partially corrects the output. Also, an error due to variations in each cell is randomly distributed through the ŁLA. A simulation with random errors distributed through the ŁLA indicated that the total error should be no greater than the error of a single cell. Our observations agree with this simulation.

A trimmed ŁLA computes fuzzy functions accurately. Error is within 1.5% of ideal, with error near 4% where the current changes value rapidly. With further modifications to the circuit, and trimmed outputs, the error should drop to less than 1% as reported by Yamakawa [1].

We close with a summary of our results and directions for future research.



(a) Untrimmed "notch" and error



(b) Optimally trimmed "notch" and error

Figure 8. ŁLA implementation of $(\neg \alpha \rightarrow \alpha) \rightarrow \neg (\alpha \rightarrow \neg \alpha)$

6. CONCLUSIONS

Łukasiewicz logic is representation complete with respect to other multiple-valued logics. Mapping sentences in Łukasiewicz logic to cellular automata leads to parallel architectures that can perform a variety of computations.

We described the architecture of an operational 31-cell CMOS VLSI ŁLA, which is regular, simple, area-efficient and implemented with analog rather than digital processing elements. Although ŁLAs are analog computers they can be made surprisingly precise (5 to 8 bits).

The prototype ŁLAs are programmed with input vectors derived from normal forms of sentences in the Łukasiewicz logic. This requires data inputs on the order of $O(2^n)$ for sentences in n implications, limits the size of the sentences that can be evaluated by a given ŁLA and increases the number of pins needed on the VLSI package. However, many data inputs are *true* or *false*, or are composed of a repeated number of variable inputs. Based on this observation we are designing ŁLAs that have external control inputs, and a restricted number of external data inputs.

ŁLA programming is an instance of the more general problem of programming analog and hybrid digital-analog computer architectures. Because ŁLA-based systems will be either analog or hybrid digital-analog computers, future research includes developing programming languages for them.

The dual logical and algebraic semantics of Łukasiewicz logic allow ŁLAs to implement expert systems, neural networks, and fuzzy logic functions. We presented schematic examples for each application, and reported the results obtained by programming the prototype ŁLA as a fuzzy function generator.

The results showed that the ŁLA implemented the notch function linearly, but with a slope that varied from that of the calculated function. This scaling was due to an arbitrary choice of a resistor in the measuring circuit. Optimally changing this resistance was calculated to yield output with a typical error of less than 2%, and a mean error less than 0.5%. Changes to the ŁLA basic cell, and MOSIS runs specifically for analog circuits, are expected to improve accuracy and increase precision.

ŁLAs present a new challenge in the design of massively parallel processors, as well as the design and programming of analog and hybrid computers. For those problems where precision may be traded for speed, ŁLAs provide an excellent solution.

ACKNOWLEDGMENTS

Gregory J. E. Rawlins' and the referees' suggestions were extremely helpful. Several long discussions with Dr. Rawlins helped improve our presentation of the ideas contained in the original version of this paper.

REFERENCES

[1] Yamakawa, T. and T. Miki. 1986. The current mode fuzzy logic integrated circuits fabricated by the standard CMOS process. *IEEE Transactions on on Computers* C-35 (2): pp. 161-167.

[2] Rumelhart, D. E., and J. L. McClelland. 1986. *Parallel Distributed Processing*. Cambridge, Massachusetts: MIT Press.

[3] Grossberg, S. 1988. *Neural Networks and Natural Intelligence*. Cambridge, Massachusetts: MIT Press.

[4] Wilkinson, R. H. 1963. A Method of Generating Functions of Several Variables Using Analog Diode Logic. *IEEE Transactions on Electronic Computers* EC-12 (2): pp. 112-128.

[5] McNaughton, R. 1951. A theorem about infinite-valued sentential logic. *Journal of Symbolic Logic* 16 pp. 1-13.

[6] Mills, J. W., and A. A. Faustini. *Lucid: An Intensional Programming Language for Łukasiewicz Logic Arrays*. (in preparation)

[7] Codd, E. F. 1968. *Cellular Automata*. New York, New York: Academic Press.

[8] Kung, H. T., and C. E. Leiserson. 1978. Systolic arrays (for VLSI). *Proceedings of Symposium on Sparse Matrices Computations*. Knoxville, Tennessee: SIAM. pp. 245-282.

[9] Alspector, J., R. B. Allen, V. Hu, and S. Satyanarayana. 1987. Stochastic learning networks and their electronic implementation. *Proceedings of Neural Information Processing Systems—Natural and Synthetic*. Denver, Colorado, November 8-12:

[10] Alspector, J., and R. B. Allen. 1987. *A neuromorphic VLSI learning system*. In *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*. Edited by P. Losleben. pp. 313-349. Cambridge, Massachusetts: MIT Press.

[11] Mead, C. 1989. *Analog VLSI and Neural Systems*. Edited by L. Conway, C. Seitz and C. Koch. VLSI System Series/Computation and Neural System Series. Reading, Massachusetts: Addison-Wesley.

[12] Belnap, N. D. 1977. *How a computer should think*. In *Contemporary Aspects of Philosophy*. Edited by G. Ryle. pp. 30-56. Boston, Massachusetts: Oriel.

[13] Leiserson, C. E. 1981. "Area-Efficient VLSI Computation." Ph.D., Dept. of Computer Science, Carnegie-Mellon University.

[14] Mills, J. W. 1989. A pipelined architecture for logic programming with a complex but single-cycle instruction set. *Proceedings of IEEE 1st International Tools for Artificial Intelligence Workshop*. Fairfax, Virginia: IEEE Computer Society Press. pp. 526-533.

[15] Celis, I., B. Cox, L.-L. Shyu, and J. W. Mills. 1989. *LIBRA bit-slice design*. Computer Science Department, Indiana University:

[16] Allen, C. M., and D. D. Givone. 1984. *The Allen-Givone implementation oriented algebra*. In *Computer Science and Multiple-Valued Logic*. Edited by D. C. Rine. pp. 268-288. Amsterdam: North-Holland.

[17] Hurst, S., L. 1986. A survey: developments in optoelectronics and its applicability to multiple-valued logic. *Proceedings of IEEE 16th Symposium on Multiple-Valued Logic*. Blacksburg, Virginia: IEEE Computer Society Press. pp. 179-188.

[18] Mattrey, R. F., D. D. Givone, and C. M. Allen. 1973. Applying multiple-valued algebra concepts to neural modeling. *Proceedings of IEEE International Symposium on Multiple-Valued Logic*. Toronto, Canada:

[19] Giles, R. 1976. Lukasiewicz logic and fuzzy set theory. *Int. J. Man-Machine Studies* 8 pp. 313-327.

[20] Yamakawa, T. 1988. High-speed fuzzy controller hardware system: The Mega-FIPS machine. *Information Sciences* 45 (2): pp. 113-128.

[21] Giles, R. 1985. A resolution logic for fuzzy reasoning. *Proceedings of IEEE 17th International Symposium on Multiple-Valued Logic*. IEEE Computer Society Press. pp. 60-67.

[22] McCulloch, W. S., and W. Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5 pp. 115-133.

[23] Kleene, S. C. 1956. *Representation of events in nerve nets and finite automata*. In *Automata Studies*. Edited by C. E. Shannon and J. McCarthy. pp. 3-41. Princeton: Princeton University Press.

[24] von Neumann, J. 1956. *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*. In *Automata Studies*. Edited by C. E. Shannon and J. McCarthy. pp. 43-99. Princeton, New Jersey: Princeton University Press.

[25] Graf, H. P., L. D. Jackel, and W. E. Hubbard. 1988. VLSI implementation of a neural network model. *IEEE Computer* 21 (3): pp. 41-49.

[26] Giles, R. 1979. *A formal system for fuzzy reasoning*. In *Fuzzy Sets and Systems* 2. pp. 233-257. North-Holland.

[27] Giles, R. 1982. Semantics for fuzzy reasoning. *Int. J. Man-Machine Studies* 17 pp. 401-415.

[28] Zadeh, L. A. 1975. Fuzzy logic and approximate reasoning. *Synthese* 30 pp. 407-428.

[29] Yamakawa, T., and H. Kabuo. 1988. A programmable fuzzifier integrated circuit—synthesis, design and fabrication. *Information Sciences* 45 (2): pp. 75-112.

[30] Lukasiewicz, J., and A. Tarski. 1930. Untersuchungen über den Aussagenkalkül. *Comptes rendus des séances de la Société des sciences et des lettres des Varsovie Classe III* (23): pp. 30-50.

[31] Gaines, B. R. 1976. Fuzzy reasoning and the logics of uncertainty. *Proceedings of IEEE 6th International Symposium on Multiple-Valued Logic*. IEEE Computer Society Press. pp. 179-188.

[32] Rescher, N. 1969. *Many-Valued Logic*. New York: McGraw-Hill.

TECHNICAL REPORT NO. 296

Lukasiewicz Logic Arrays

by

Jonathan W. Mills, M. Gordon Beavers and Charles A. Daffinger

Revised: March 1990

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
Bloomington, Indiana 47405-4101

Łukasiewicz Logic Arrays

Jonathan Wayne Mills
M. Gordon Beavers
Charles A. Daffinger

Indiana University
Bloomington, Indiana 47405

Abstract

Łukasiewicz logic arrays (ŁLAs) are massively parallel analog computers organized as binary trees of identical processing elements performing either implication (→), negated implication (↛) or both. We have designed and built a working 31-cell CMOS VLSI ŁLA whose cells perform implication (→). In this paper we discuss the representation completeness of Łukasiewicz logic with respect to other multiple-valued logics, describe the architecture of the prototype ŁLA, its relationship to cellular automata and its VLSI implementation, show how the prototype ŁLA is programmed, and report on results obtained by programming the prototype ŁLA as a fuzzy function generator. Because ŁLAs have both an algebraic and a logical operational semantics, they can be used to implément approximate reasoning systems, including expert systems and neural networks.

1. INTRODUCTION

Łukasiewicz logic arrays (ŁLAs) are massively parallel analog computers. They are organized as binary trees of identical processing elements (called PEs, or cells), each PE performing either Łukasiewicz implication (→), negated implication (↛) or both.

We have designed and built a working 31-cell CMOS VLSI ŁLA whose cells perform implication (→). In this paper we discuss the representation completeness of Łukasiewicz logic with respect to other multiple-valued logics, describe the architecture of the prototype ŁLA, its relationship to cellular automata and its VLSI implementation, show how the prototype ŁLA is programmed, and report on results obtained by programming the prototype ŁLA as a fuzzy function generator.

The success of the prototype has encouraged us to continue research in the design and application of ŁLAs. During this research we have observed that ŁLAs offer advantages as massively parallel analog computers.

1.1 ADVANTAGES

ŁLAs are regular VLSI architectures. The VLSI implementation of ŁLAs is simple and area-efficient because they are derived from cellular automata, and implemented with analog rather than digital processing elements. Although ŁLAs are analog computers they can be made surprisingly precise (5 to 8 bits), due to the simplicity of their processing elements and the accuracy of VLSI process technology.

ŁLAs are inductive architectures, which means that they can be expanded by adding more processing elements without redesigning the interconnection network. While small ŁLAs can be used as circuit components, large ŁLAs can be used as massively parallel computers. Larger ŁLAs can be created by cascading individual ŁLAs.

The general-purpose nature of ŁLAs is theoretically well-founded. Multiple-valued logics used in computational networks are capable of both symbolic and algebraic computation. ŁLAs can implement fuzzy inference and expert systems [1], neural networks [2, 3], and algebraic functions [4, 5]. Viewed as circuit components, ŁLAs are the multiple-valued logic equivalent of programmable logic arrays (PLAs) for Boolean logic.

1.2 DISADVANTAGES

Of course, ŁLAs are not ideal analog processors, but we are working to reduce their drawbacks.

The prototype ŁLAs are programmed using normal forms of sentences in the Łukasiewicz logic. This introduces data inputs on the order of $O(2^n)$ for sentences in n implications, and limits the size of the sentences that can be evaluated by a given ŁLA. Using the normal form also increases the number of pins needed on the VLSI package, far beyond the number available even in the foreseeable future. However, many data inputs are *true* or *false*, or are composed of a repeated number of variable inputs. Based on this observation we are designing ŁLAs that have external control inputs, and a restricted number of external data inputs. The data inputs are replicated and selected internally at each input of the processor array according to the externally applied control inputs.

Because ŁLAs are a new form of computational engine their use is still being studied. We have only a basic understanding of the programming methodology for ŁLAs. For example, the theoretical applicability of ŁLAs as neural networks does not immediately lead to the construction of algorithms for back-propagation.

ŁLA programming is an instance of the more general problem of programming analog and hybrid digital-analog computer architectures. Research in this area stopped about 1970 due to the dominance of digital computers. Because ŁLA-based systems will be either analog or hybrid digital-analog computers we must develop programming languages for them. Mills and Faustini [6] have proposed a language for ŁLA-based systems, but its operational semantics are still only partly defined. Completion of the semantics will require a more exact characterization of the dynamic behavior of ŁLAs, particularly ŁLAs with cyclic interconnections.

The next section describes Łukasiewicz logic and its representation completeness relative to the class of multiple-valued logics whose valuation functions can be defined in terms of +, −, *min* and *max*.

2. THE MULTIPLE-VALUED LOGICS CLASSES $L_{[0,1]}$ AND $L_{\{+,-,\wedge,\vee\}}$

(Lukasiewicz and Tarski 1930) contains a compendium of the results of investigation into multiple-valued logics obtained by Lukasiewicz and his students in the 1920's. Following the initial efforts of Lukasiewicz and Post other multiple-valued logics were developed, both discrete and continuous. Summaries of these logics can be found in (Rescher 1969) and (Gaines 1976). Most of these logics belong to $L_{[0,1]}$ which is given by:

Definition 1. *A logic L is a member of the class $L_{[0,1]}$ iff there is a logical matrix M appropriate for L with $\mathrm{M} = <\mathrm{P}, D>$ where P is a non-empty algebra whose carrier set is a subset of the real number range [0,1], with D, the set of designated elements, a non-empty proper subset of the carrier set.*

The class $L_{[0,1]}$ can be further restricted to yield a class of logics whose valuation functions for the connectives can be expressed in terms of addition, subtraction, maximum, and minimum alone. This class will be denoted by $L_{\{+,-,\wedge,\vee\}}$.

Definition 2. *A logic L in $L_{[0,1]}$ is in $L_{\{+,-,\wedge,\vee\}}$ iff all sentences of L can be evaluated using only the operators $+, -, max, min$ on the values of the atomic sentences of L.*

 To appear in the Proceedings of the IEEE 20th International Symposium on Multiple-Valued Logic

The importance of the class $L_{\{+,-,\wedge,\vee\}}$ is that it contains only those logics whose sentences can be easily evaluated by analog circuits using electrical current to represent the values of logical variables. The valuation functions of these logics can be implemented by adding and subtracting electrical currents – simple operations for electronic devices: and by utilizing Ohm's law, Kirchoff's law and the law of conservation of energy to implicitly implement the operations *max* and *min* for electrical currents using the physical properties of the circuit. Furthermore, this class contains logics, such as fuzzy logic, whose significance to the fields of approximate reasoning and artificial intelligence is well established.

2.1 REPRESENTATION COMPLETENESS OF LUKASIEWICZ LOGIC RELATIVE TO $L_{\{+,-,\wedge,\vee\}}$

That Lukasiewicz logics are members of the class $L_{\{+,-,\wedge,\vee\}}$ follows from:

Definition 3. *Ł is a Lukasiewicz logic if Ł has a model* $M = < A, D >$ *where* $A = < S, \neg, \rightarrow >$ *and S is a subset of* $[0,1]$ *such that:*

1. $1 \in S$,
2. *If* $x, y \in S$ *then* $min(1, 1 - x + y) \in S$, *and* $1 - x \in S$,

where $x \rightarrow y$ *and* $\neg x$ *are evaluated as* $1 - x + y$ *and* $1 - x$ *respectively.*

If $S = [0,1]$ we get the classical propositional calculus. If S has n elements then we get the n-valued Lukasiewicz propositional calculus $Ł_n$. If the cardinality of S is $\aleph_0$ or $\aleph_1$ we get $Ł_{\aleph_0}$ or $Ł_{\aleph_1}$ which happen to have the same set of theorems. We designate $Ł_{\aleph_0}$ by Ł and take S for Ł to be the set of rational numbers between 0 and 1.
(Giles 1976) shows that Zadeh's seminal work on fuzzy set theory is closely related to Lukasiewicz logic. That Ł is representation complete with respect to the class of logics $L_{\{+,-,\wedge,\vee\}}$ follows from the fact that the evaluation of formulae in Ł has the following properties:

1. $v(\neg(\phi)) = v(\phi \rightarrow 0)$.
2. $max(v(\phi), v(\psi)) = v((\phi \rightarrow \psi) \rightarrow \psi)$.
3. $min(v(\phi), v(\psi)) = v(\neg((\neg\phi \rightarrow \neg\psi) \rightarrow \neg\psi))$
4. $min(1, v(\phi) + v(\psi)) = v(\neg\phi \rightarrow \psi)$

With the ability to perform these calculations, we have the first step in our justification of the LLA as a fundamental circuit for approximate reasoning.

2.2 McNAUGHTON'S THEOREM

McNaughton's theorem allows us to use Ł at different levels of abstraction, in particular as a classifier for elements of fuzzy sets. By showing that valuation functions for connectives in sentences in Ł are equivalent piecewise to polynomials of degree one that map the hyperspace $[0,1]^n$ into the interval $[0,1]$, the capability of building fuzzy pattern recognizers is provided. Thus a series of sentences in Ł defines the polytope of some simply connected solid in hyperspace of degree n. This allows us to express arbitrarily complex membership relations in logical form; in a VLSI circuit we define the polytope with a sentence from Ł, which is converted to the normal form (derived in section 4) of the sentence representable by one or more LLA circuits. The LLA circuit may be "programmed" to deal with variants of the original sentence by assigning incoming data to specific circuit inputs.

Theorem 1. (McNaughton 1951) *Let* $u_1, ..., u_n$ *be numerical variables and* $x_1, ..., x_n$ *be propositional variables. For a function* $f(u_1, ..., u_n)$ *there is a logical formula* ϕ *of Ł such that* $f(u_1, ..., u_n) = v(\phi(x_1, ..., x_n))$ *iff*

(i) f *is continuous over* $[0,1]^n$ *and* $Range(f) \subseteq [0,1]$, *and*
(ii) there is a finite number t of distinct polynomials $\pi_1, ..., \pi_t$ *each of the form*

$$\pi_j = b_j + m_{1,j}u_1 + \cdots + m_{n,j}u_n$$

with b_j, $m_{i,j}$ *integers such that for every* $< u_1, ..., u_n >$ *there is a j such that*

$$f(u_1, ..., u_n) = \pi_j(u_1, ..., u_n).$$

Next, homogeneous, heterogeneous and logical cellular automata are defined, and on this basis a Łukasiewicz logic array is developed that implements implication for Ł.

3. ARCHITECTURE

3.1 DESIGN

Łukasiewicz logic arrays resulted from research into cellular automata as parallel architectures for logic programming.[1] Cellular automata are of particular interest because they lead to area-efficient VLSI architectures. Such architectures are implemented as regular arrays of processing elements which communicate the results of their computation locally. They are derived by instantiating a portion of a cellular automaton as a VLSI circuit. The structure and function of the circuit arises from the definition of a cellular automaton:

Definition 4. [7] A cellular automaton C is defined by the quadruple (S, g, h^t, *ϕ) where:*

- S — *is a two-dimensional cellular space defined by the set of cells* $\alpha \in I \times I$ *where I denotes the set of integers.*
- g — *is a neighborhood function mapping* $S \rightarrow 2^S$ *such that* $g(\alpha)$ *is a set defining the cells in the neighborhood of* α *relative to* α. *Typically* α *is a member of its own neighborhood.*
- h^t — *is a neighborhood state function at some time t. Values of cells in the neighborhood of* α *at time t are obtained by applying* h^t *to the neighborhood* $g(\alpha)$. *The successive states of* α *at times* $\{t_0, t_1, t_2, ...\}$ *can be defined by the composition* $f \circ h^t \circ g(\alpha) = v^{t+1}(\alpha)$.
- ϕ — *is a finite automaton replicated in each cell of S and defined by the triple* (V, v_0, f). *V is the set of states possible for each cell,* v_0 *a distinguished quiescent state, and f a transition function mapping n-tuples of elements of V into V. The transition function f is constrained to preserve quiescence locally by requiring* $f(v_0, v_0, ..., v_0) = v_0$.

A cellular automaton is *homogeneous* if the neighborhood function and the finite automaton are identical for all cells in the cellular space S at all times *t*, otherwise the cellular automaton is *heterogeneous*.

Heterogeneous cellular automata model a wide variety of parallel computational devices. Examples include the systolic architectures of Kung and Lieserson [8], the stochastic neural machines of Alspector et. al. [9, 10] and the analog VLSI computers of Mead [11].

Ideal Łukasiewicz logic arrays (ŁLAs) are heterogeneous cellular automata that implement a denumerably infinite sentence schema of Ł. The sentence schema of Ł and the cellular automaton C are related by requiring the logical variables of Ł to correspond to cells in the cellular space S, the structure of the sentence schema to correspond to the neighborhood function g, and the connectives of Ł to correspond to the transition function f of ϕ. Ł is therefore a logic in the sense used by Belnap — an *organon*, or a tool for inference — and not a formal axiomatic theory [12].

Real Łukasiewicz logic arrays are derived by restricting the denumerably infinite sentence schema of Ł to a finite sentence schema, and implementing the finite cellular automaton that results as a direct correspondence architecture. The structure of the resulting ŁLA is dependent on its interconnection network. The prototype ŁLA uses an H-tree network whose nodes are the processing elements corresponding to the connectives in the finite sentence schema. The H-tree network was selected for its efficient use of area on a VLSI circuit, as first proposed by Leiserson [13].

1 In the general sense of "programming with logic" rather than the restricted sense of implementing Prolog.

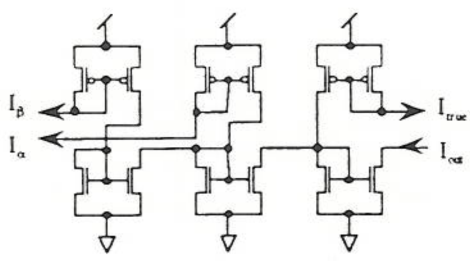
3.2 VLSI IMPLEMENTATION

Łukasiewicz logic arrays are implemented with analog processing elements. A cell in the ŁLA is implemented as an analog current-mode device performing addition, subtraction, *min* and *max* on currents. Addition and subtraction are done instantaneously, though the circuit needs a short time to stabilize.[2] Early in our work we learned of a series of fuzzy functions implemented by a basic logic cell [1]. The circuits which implement these functions also implement the algebraic valuation functions for Ł. For our purposes the most useful of Yamakawa's circuits are implication (→), which computes $min(1, 1 - \alpha + \beta)$, and bounded difference, which computes $max(0, \alpha - \beta)$. Algebraically reducing the expression for negated implication ($\neg(\alpha \rightarrow \beta)$, or $\alpha \not\rightarrow \beta$) from $(1 - min(1, 1 - \alpha + \beta))$ yields $max(0, \alpha - \beta)$, showing that it is equivalent to the bounded difference.

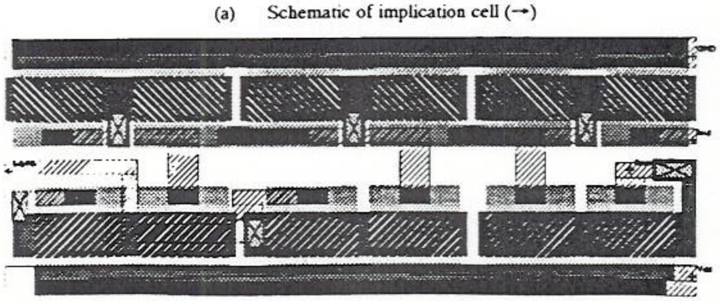
The design uses Kirchhoff's laws to sum currents at points within the ŁLA cell. To ensure that the varying current drains of adjacent cells do not affect the computation of their predecessors, as well as guaranteeing a proper input voltage, each cell is isolated by a set of current mirrors. MOS FET current mirrors have very good accuracy in making any number of copies of a given input current without placing a variable drain on that input current.

The basic cell consists of six current mirrors, and performs Łukasiewicz implication (→). A cell has two inputs and a single output, and is designed to be tiled in an H-tree (Figure 1a). The basic cell uses 11 transistors, and is 35μ by 114μ using the 2μ SCPE technology provided by the MOSIS fabrication service (Figure 1b). Basic cells are combined in an H-tree to form the ŁLA (Figure 1c).

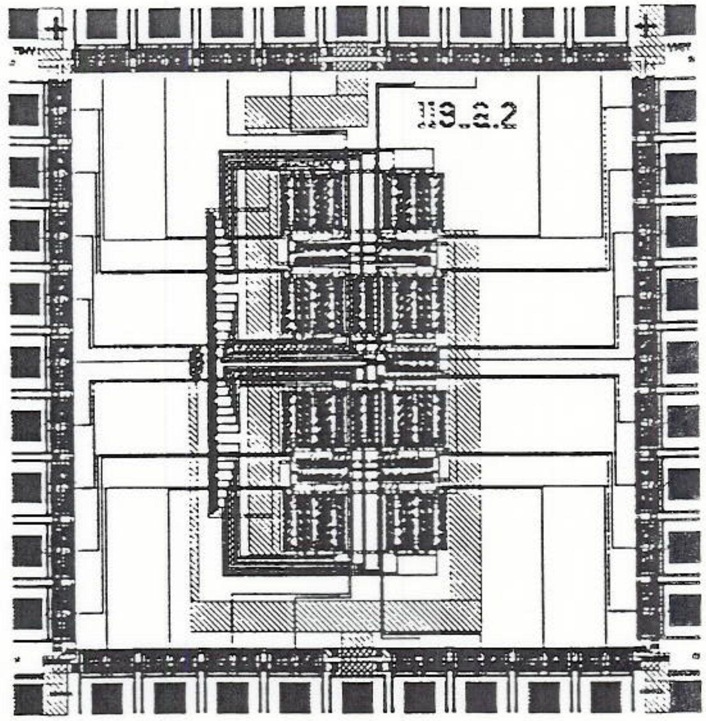
The operating range for the ŁLA cell varies from 0 to 7 volts with input and output currents varying from 0 to 20 microamperes (μA). Within this range the accuracy of the ŁLA is affected by three sources of error. The first is steady-state error, which is dependent on the actual dimensions of the transistors and other process parameters for a particular MOSIS run. The second source of error is temperature dependent, and varies as the temperature changes over long periods of time. As long as the temperature of the system in which the ŁLA is placed varies uniformly this error can be ignored. The third source of error is transient error which arises when large current swings occur in the inputs of the ŁLA, and lasts until the cell has stabilized.



(a) Schematic of implication cell (→)



(b) Layout of implication cell (→)



(c) Layout of a 31-cell 5-level ŁLA

Figure 1. Heterogeneous ŁLA in implication (→)

SPICE simulations indicated that steady-state error is well-behaved, and remains within 1.5% mean and 4% maximum for small cells, growing slowly as the depth of the ŁLA is increased. Our observations agree with the simulation.

The transient error is dependent upon how chaotic the inputs of the circuit are. This is related to the number of inputs that change during a sampling interval, the amount by which they change, and the level of current used for the *true*, or maximum value. Selecting a high value for *true* increases the precision of the ŁLA but at a price: larger current swings will require a longer settling time, and produce a slower circuit.

Choosing an analog processing element yields several advantages. Because the ŁLA is a current-mode circuit it has a precision which is not achievable with an equivalently-sized voltage-mode circuit. Although Ł is infinitely valued, in practice only $Ł_2$ through $Ł_{256}$ can be implemented due to device error and the resolution of our measuring devices. The output error measured for the prototype ŁLAs is in the range of 0.25% to 2%. This gives an information density ranging from 5.6 to 8.6 bits, or approximately 50 to 400 discreet values per ŁLA. This is a useful precision for approximate reasoning systems.

The processing elements are simple, performing only Łukasiewicz implication (→) to evaluate the sentences in Ł defined by the schema. Processing elements need only two input wires and one output wire because they use analog values. Thus, the bus structure of the ŁLA is also area-efficient.

The total area used by an ŁLA is much less than the area required for an equivalent digital processor. This is based on the number of transistors needed to implement the digital processor's arithmetic logic unit (ALU) and register file, but not its control and bus interface circuitry. If each processing element has eight bits of precision, then the LIBRA digital ALU [14, 15] uses 935 times more transistors and is 1,020 times larger than the basic cell of the ŁLA (Table 1).

2 Simulations have been conducted on a timescale of microseconds, with the response of the circuit to a change of inputs instantaneous on that scale.

Table 1. Comparison of the Łukasiewicz logic array to a digital Prolog processor

	ŁLA	LIBRA	Increase
Transistors	11	10,288	935×
Area	$6{,}272\mu^2$	$6.4 \times 10^6\mu^2$	1,020×

However, one drawback to an area-efficient circuit is that it is limited by the number of pins available on existing VLSI circuit packages. Although an array of 1024 Łukasiewicz implication (→) cells could easily fit onto a 4500µ × 2300µ chip, it would require 2048 input pins and 1 output pin. This is 1,921 more pins than are available on a 128 pin-grid array package. Our research has shown that many functions implemented with ŁLAs will have more than half of their inputs tied to *true* or *false*. For these functions Ł)LAs can be built that use a programmable interconnection network to route internally replicated *true* and *false* inputs to the PE array. Data inputs also tend to be used more than once, so they could be internally replicated and routed, too. This approach allows large ŁLAs to fit into existing VLSI packages.

The ŁLAs described here resemble 1960's-era analog and hybrid digital-analog computers. This leads to the view of ŁLA programming as an instance of the more general problem of programming analog and hybrid computer architectures. We develop a low-level ŁLA programming methodology in the following section.

4. PROGRAMMING IN ŁUKASIEWICZ LOGIC

ŁLAs are programmed at the lowest level by fixing an interconnection network for the inputs, and presenting inputs that are either *true*, *false*, or variable. Because it is not practical to build an ŁLA for each sentence in Ł, it is necessary to develop a normal form that maps arbitrary sentences onto some general ŁLA.

4.1 A BALANCED NORMAL FORM FOR Ł

The prototype ŁLA is structured as a binary tree whose nodes are connectives, and whose leaves are logical variables. Most sentences in Ł do not map directly to this schema, but must be transformed to equivalent sentences which do. This general form of a sentence in Ł is the balanced normal form in implication, with explicit negation possible anywhere in the sentence.

Definition 5. *A sentence in Ł is in balanced normal form in implication if there exists some designated implication in the sentence, starting at which a binary tree of implications can be extracted, and for which at each non-leaf node in the tree the number of implications and logical variables in each subtree rooted at that node is equal.*

Theorem 2. *Any sentence in Ł can be rewritten to an equivalent sentence in balanced normal form in implication.*

Production of this balanced normal form can be viewed as an inverse operation of the minimization of Allen and Givone [16]. The circuit implements balanced normal form sentences in Ł because it is structured as an H-tree. The use of a binary tree to realize n-input R-valued functions for multiple-valued logic circuits was described by [17].

4.2 NEGATION-FREE NORMAL FORM

The next step toward developing a useful normal form is the transfer of negation from an arbitrary point in any sentence, moving it to either the root or one or more leaves in the binary tree of connectives and logical variables.

Unfortunately, we suspect that no normal form for Łukasiewicz logic exists with negation moved to either leaf or root implications. But it is just as useful from a computer architect's point of view to leave the negation in place as long as the negated expression $\neg\alpha$ can be re-written to an equivalent form that does not use negation explicitly, namely $\alpha \rightarrow$ *false*. A clause expressed in only one connective, while textually more complex, may be mapped to smaller and simpler physical devices that perform negation using data inputs alone.

The balanced negation-free normal form is obtained by removing negation from any sentence of Ł by simplification where possible, or by rewriting negation as $\alpha \rightarrow$ *false* otherwise. To define the balanced negation-free normal form we first define a *negation-free normal form* as follows.

Definition 6. *A sentence in Ł is in negation-free normal form iff it is expressed only in implication, and contains some designated connective such that a binary tree of connectives can be constructed whose root is the designated connective and whose leaves are logical variables in the clause.*

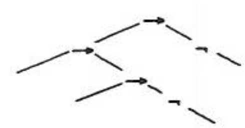
To continue the transformation the concept of the *weight* of a tree must be defined (it was implicit in Definition 5 of the balanced normal form). From this it is a short step to the definition of the balanced negation-free normal form *(BNF normal form)* and an equivalence theorem.

Definition 7. *The weight of a tree is the number of connectives and logical variables contained in the tree.*

Definition 8. *A clause is in BNF normal form iff it is in negation-free normal form, and at each non-leaf node in the tree the weights of each subtree are equal.*

Theorem 3. *Any sentence in Ł can be transformed to BNF normal form.*

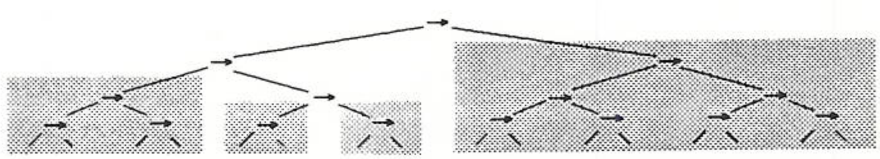
The proofs of Theorems 2 and 3 are omitted, but an example provides their substance. Consider the transformation of an arbitrary sentence in Ł to BNF normal form. The sentence is unbalanced initially, and contains negation (Figure 2).



$(\alpha \rightarrow (\beta \rightarrow \neg\gamma)) \rightarrow \neg\delta$

Figure 2. Unbalanced sentence in Ł

The resultant BNF normal form to which it is transformed is shown next (Figure 3). Although the textual form of the sentence is more complex, the BNF normal form uses cells of the ŁLA that the first form would have left unused. These "extra" inputs and implications can be used to adjust the constraints under which the sentence is true.



$(((T \rightarrow T) \rightarrow (T \rightarrow \alpha)) \rightarrow ((T \rightarrow \beta) \rightarrow (\gamma \rightarrow F))) \rightarrow (((T \rightarrow T) \rightarrow (T \rightarrow \delta)) \rightarrow ((T \rightarrow T) \rightarrow (T \rightarrow F)))$

Figure 3. BNF normal form of sentence in Figure 2

4.3 COMPLEXITY OF THE NORMAL FORM

The proof that the number of logical variables and connectives in the normal form is of complexity $O(2^n)$ where n is the minimum height of the trees formed from an arbitrary sentence of Ł is outlined: simplify all negations, then treat any negation remaining as a node in the tree; generate a set of trees by successively designating each implication in the sentence as the root connective, then select n equal to the minimum of the height of all generated trees. The number of inputs is at most 2^{n+1}, and the number of nodes in the tree is 2^n. Although the presence of exponential complexity in both normal forms is disturbing, some optimizations are possible. For example, if a sentence in Ł is transformed to BNF normal form, many of the inputs on the original degenerate branch are either *true* or *false*. When a normal form is so large that it spans multiple VLSI circuits, then it is possible to remove the *true* and *false* inputs by supplying the single value instead of computing it with a series of LLAs.

5. APPLICATIONS OF ŁLAS

Łukasiewicz logic arrays were first proposed to evaluate sentences in Ł, but because Łukasiewicz logic describes other forms of approximate reasoning, Ł)LAs are useful for a variety of applications. The dual logical and algebraic semantics of Ł allow ŁLAs to implement expert systems, neural networks [10, 18], and fuzzy computers [19, 20]. We present schematic examples for each application, and report the results obtained by programming the prototype ŁLA as a fuzzy function generator.

5.1 EXPERT SYSTEMS

ŁLAs implement expert systems by mapping membership functions to processing elements at lower levels in the array, and rules to processing elements higher in the array. A rule is a single tree that is true or false to a degree that depends on its inputs. Rules can be designed that do not fire unless their inputs reach a desired confidence level (Figure 4).

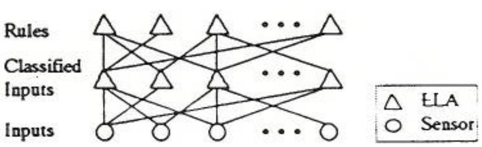


Figure 4. ŁLA implementation of expert system

A single Łukasiewicz logic array can implement a simple expert system, which may be used to embed limited "intelligence" in individual sensors. Within the array some processing elements operate at a low level of abstraction, evaluating membership in a fuzzy set, while other processing elements operate at a higher level of abstraction, implementing a rule for that sensor. The rule's operation may vary based on control inputs to the ŁLA. The expert system evaluates its sensor's input, firing the rule if the confidence factor is exceeded.

5.2 NEURAL NETWORKS

McNaughton's theorem (see Section 2) and Giles' Logic of Assertions [21] relate sentences in Łukasiewicz logic to piecewise-linear functions and the theory of convex analysis. This is the functional domain of pattern recognizers and classifiers (Figure 5), which has encouraged us to investigate neural networks implemented with ŁLAs.

Early models of nerve nets were described by McCulloch and Pitts [22]. Kleene and von Neumann anticipated much of the present-day work in neural networks, offering theoretical descriptions of the events representable in neural networks [23], and the creation of reliable computing systems from unreliable components [24].

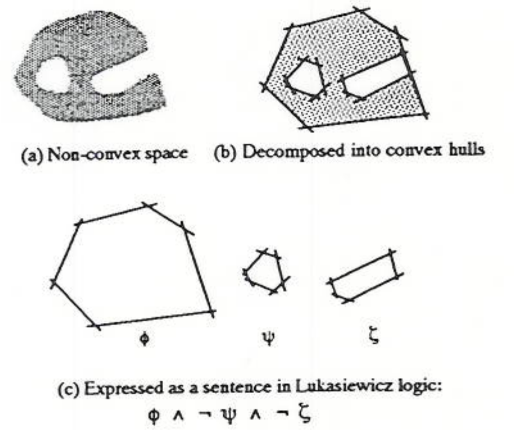


Figure 5. Relationship of non-convex space to Łukasiewicz logic

The evaluation formula for Łukasiewicz implication shows how it may be used to construct a very simple "neuron." In the expression $min(1, 1 - \alpha + \beta)$, α is an inhibitory input that lowers the "firing rate", or truth value. β is an excitatory input that increases the "firing rate." Recursively connecting several implication cells produces a "neuron" with a variable threshold (Figure 6a). Summation units can also be devised (Figure 6b).

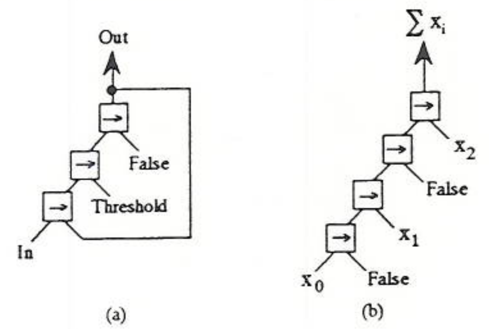


Figure 6. Implementing a "neuron" and summing element with an ŁLA

Simulations of interconnected ŁLA "neurons" and summing elements show that they have the basic properties needed to construct a neural network. The behavior of a "neuron" can be changed by modifying its threshold. For example, the slight delay before the second output pulse in the simulation is due to an intermediate "neuron" (Figure 7).

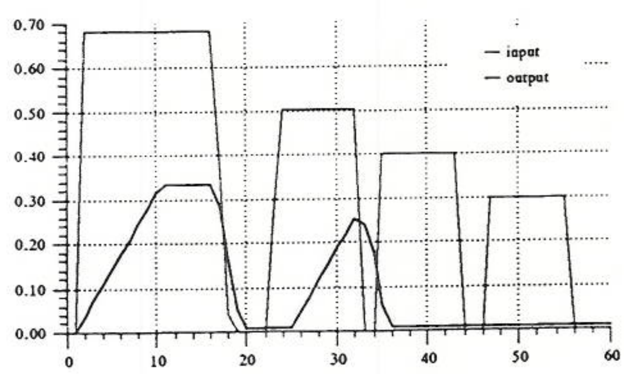


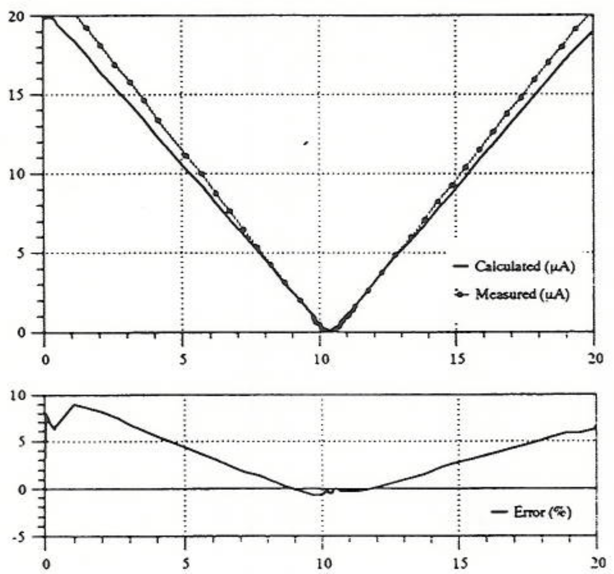
Figure 7. Simulation of interconnected ŁLA "neurons"

We are now working to construct a trainable neural network from these basic components. The initial version will be a hybrid digital-analog system, similar to those of Alspector and Graf [9, 10, 25]. However, we hope to devise analog-only systems using double-poly capacitors as storage elements for weights.

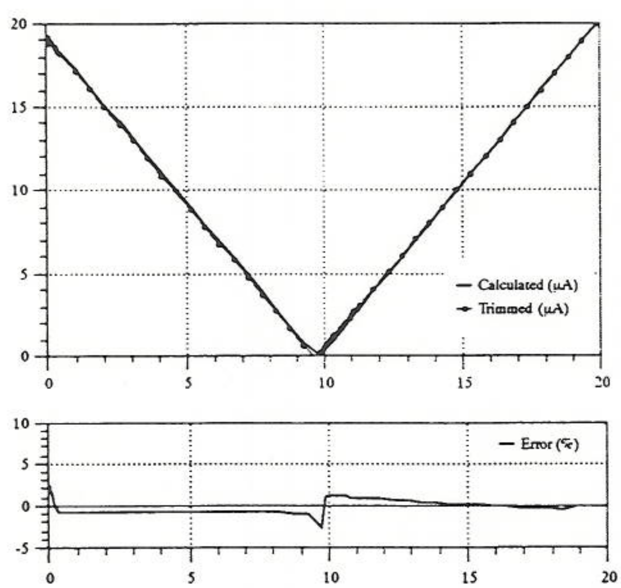
5.3 FUZZY LOGIC

Łukasiewicz logic is closely related to fuzzy logic [19, 21, 26, 27, 28]. Yamakawa shows designs for fuzzy inference engines and expert systems which may be embedded in Łukasiewicz logic arrays [1, 20, 29]. We have used the prototype ŁLA to compute fuzzy membership functions, and present the results obtained along with observations on the error measured using the prototype ŁLA.

An interesting fuzzy membership function is the "notch", defined by the expression $(\neg \alpha \rightarrow \alpha) \rightarrow \neg (\alpha \rightarrow \neg \alpha)$. This membership function



(a) Untrimmed "notch" and error



(b) Optimally trimmed "notch" and error

Figure 8. ŁLA implementation of $(\neg \alpha \rightarrow \alpha) \rightarrow \neg (\alpha \rightarrow \neg \alpha)$

was programmed into the 31-cell ŁLA as the following 32-element vector:

$$(T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,\alpha,F,T,\alpha,T,\alpha,\alpha,F,T,T,T,F).$$

The output of the ŁLA was measured over the operating range of 0 to 20 µAmperes (µA) by varying α. The membership function was also calculated after adjusting the evaluation function for Łukasiewicz implication to the operating range of the circuit (Figure 8a).

The result of this experiment showed that the ŁLA implemented the notch function linearly, but with a slope that varied from that of the calculated function. This scaling is due to an arbitrary choice of a resistor in the measuring circuit. By changing this resistance ("trimming"), the output can be adjusted to produce a much closer fit. We have calculated that an optimally trimmed ŁLA would have a typical error of less than 2%, and a mean error less than 0.5% (Figure 8b).

Several factors contribute to the observed error. The *true* reference for implication (→) only partially corrects the output. Also, an error due to variations in each cell is randomly distributed through the ŁLA. A simulation with random errors distributed through the ŁLA indicated that the total error should be no greater than the error of a single cell. Our observations agree with this simulation.

A trimmed ŁLA computes fuzzy functions accurately. Error is within 1.5% of ideal, with error near 4% where the current changes value rapidly. With further modifications to the circuit, and trimmed outputs, the error should drop to less than 1% as reported by Yamakawa [1].

We close with a summary of our results and directions for future research.

6. CONCLUSIONS

Łukasiewicz logic is representation complete with respect to other multiple-valued logics. Mapping sentences in Łukasiewicz logic to cellular automata leads to parallel architectures that can perform a variety of computations.

We described the architecture of an operational 31-cell CMOS VLSI ŁLA, which is regular, simple, area-efficient and implemented with analog rather than digital processing elements. Although ŁLAs are analog computers they can be made surprisingly precise (5 to 8 bits).

The prototype ŁLAs are programmed with input vectors derived from normal forms of sentences in the Łukasiewicz logic. This requires data inputs on the order of $O(2^n)$ for sentences in n implications, limits the size of the sentences that can be evaluated by a given ŁLA and increases the number of pins needed on the VLSI package. However, many data inputs are *true* or *false*, or are composed of a repeated number of variable inputs. Based on this observation we are designing ŁLAs that have external control inputs, and a restricted number of external data inputs.

ŁLA programming is an instance of the more general problem of programming analog and hybrid digital-analog computer architectures. Because ŁLA-based systems will be either analog or hybrid digital-analog computers, future research includes developing programming languages for them.

The dual logical and algebraic semantics of Łukasiewicz logic allow ŁLAs to implement expert systems, neural networks, and fuzzy logic functions. We presented schematic examples for each application, and reported the results obtained by programming the prototype ŁLA as a fuzzy function generator.

The results showed that the ŁLA implemented the notch function linearly, but with a slope that varied from that of the calculated function. This scaling was due to an arbitrary choice of a resistor in the measuring circuit. Optimally changing this resistance was calculated to yield output with a typical error of less than 2%, and a mean error less than 0.5%. Changes to the ŁLA basic cell, and MOSIS runs specifically for analog circuits, are expected to improve accuracy and increase precision.

ŁLAs present a new challenge in the design of massively parallel processors, as well as the design and programming of analog and hybrid computers. For those problems where precision may be traded for speed, ŁLAs provide an excellent solution.

ACKNOWLEDGMENTS

Gregory J. E. Rawlins' and the referees' suggestions were extremely helpful. Several long discussions with Dr. Rawlins helped improve our presentation of the ideas contained in the original version of this paper.

REFERENCES

[1] Yamakawa, T. and T. Miki. 1986. The current mode fuzzy logic integrated circuits fabricated by the standard CMOS process. *IEEE Transactions on on Computers* C-35 (2): pp. 161-167.

[2] Rumelhart, D. E., and J. L. McClelland. 1986. *Parallel Distributed Processing*. Cambridge, Massachusetts: MIT Press.

[3] Grossberg, S. 1988. *Neural Networks and Natural Intelligence*. Cambridge, Massachusetts: MIT Press.

[4] Wilkinson, R. H. 1963. A Method of Generating Functions of Several Variables Using Analog Diode Logic. *IEEE Transactions on Electronic Computers* EC-12 (2): pp. 112-128.

[5] McNaughton, R. 1951. A theorem about infinite-valued sentential logic. *Journal of Symbolic Logic* 16 pp. 1-13.

[6] Mills, J. W., and A. A. Faustini. *Łucid: An Intensional Programming Language for Łukasiewicz Logic Arrays*. (in preparation)

[7] Codd, E. F. 1968. *Cellular Automata*. New York, New York: Academic Press.

[8] Kung, H. T., and C. E. Leiserson. 1978. Systolic arrays (for VLSI). *Proceedings of Symposium on Sparse Matrices Computations*. Knoxville, Tennessee: SIAM. pp. 245-282.

[9] Alspector, J., R. B. Allen, V. Hu, and S. Satyanarayana. 1987. Stochastic learning networks and their electronic implementation. *Proceedings of Neural Information Processing Systems—Natural and Synthetic*. Denver, Colorado, November 8-12:

[10] Alspector, J., and R. B. Allen. 1987. *A neuromorphic VLSI learning system*. In *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*. Edited by P. Losleben. pp. 313-349. Cambridge, Massachusetts: MIT Press.

[11] Mead, C. 1989. *Analog VLSI and Neural Systems*. Edited by L. Conway, C. Seitz and C. Koch. VLSI System Series/Computation and Neural System Series. Reading, Massachusetts: Addison-Wesley.

[12] Belnap, N. D. 1977. *How a computer should think*. In *Contemporary Aspects of Philosophy*. Edited by G. Ryle. pp. 30-56. Boston, Massachusetts: Oriel.

[13] Leiserson, C. E. 1981. "Area-Efficient VLSI Computation." Ph.D., Dept. of Computer Science, Carnegie-Mellon University.

[14] Mills, J. W. 1989. A pipelined architecture for logic programming with a complex but single-cycle instruction set. *Proceedings of IEEE 1st International Tools for Artificial Intelligence Workshop*. Fairfax, Virginia: IEEE Computer Society Press. pp. 526-533.

[15] Celis, I., B. Cox, L.-L. Shyu, and J. W. Mills. 1989. *LIBRA bit-slice design*. Computer Science Department, Indiana University:

[16] Allen, C. M., and D. D. Givone. 1984. *The Allen-Givone implementation oriented algebra*. In *Computer Science and Multiple-Valued Logic*. Edited by D. C. Rine. pp. 268-288. Amsterdam: North-Holland.

[17] Hurst, S., L. 1986. A survey: developments in optoelectronics and its applicability to multiple-valued logic. *Proceedings of IEEE 16th Symposium on Multiple-Valued Logic*. Blacksburg, Virginia: IEEE Computer Society Press. pp. 179-188.

[18] Mattrey, R. F., D. D. Givone, and C. M. Allen. 1973. Applying multiple-valued algebra concepts to neural modeling. *Proceedings of IEEE International Symposium on Multiple-Valued Logic*. Toronto, Canada:

[19] Giles, R. 1976. Lukasiewicz logic and fuzzy set theory. *Int. J. Man-Machine Studies* 8 pp. 313-327.

[20] Yamakawa, T. 1988. High-speed fuzzy controller hardware system: The Mega-FIPS machine. *Information Sciences* 45 (2): pp. 113-128.

[21] Giles, R. 1985. A resolution logic for fuzzy reasoning. *Proceedings of IEEE 17th International Symposium on Multiple-Valued Logic*. IEEE Computer Society Press. pp. 60-67.

[22] McCulloch, W. S., and W. Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5 pp. 115-133.

[23] Kleene, S. C. 1956. *Representation of events in nerve nets and finite automata*. In *Automata Studies*. Edited by C. E. Shannon and J. McCarthy. pp. 3-41. Princeton: Princeton University Press.

[24] von Neumann, J. 1956. *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*. In *Automata Studies*. Edited by C. E. Shannon and J. McCarthy. pp. 43-99. Princeton, New Jersey: Princeton University Press.

[25] Graf, H. P., L. D. Jackel, and W. E. Hubbard. 1988. VLSI implementation of a neural network model. *IEEE Computer* 21 (3): pp. 41-49.

[26] Giles, R. 1979. *A formal system for fuzzy reasoning*. In *Fuzzy Sets and Systems 2*. pp. 233-257. North-Holland.

[27] Giles, R. 1982. Semantics for fuzzy reasoning. *Int. J. Man-Machine Studies* 17 pp. 401-415.

[28] Zadeh, L. A. 1975. Fuzzy logic and approximate reasoning. *Synthese* 30 pp. 407-428.

[29] Yamakawa, T., and H. Kabuo. 1988. A programmable fuzzifier integrated circuit—synthesis, design and fabrication. *Information Sciences* 45 (2): pp. 75-112.

[30] Lukasiewicz, J., and A. Tarski. 1930. Untersuchungen über den Aussagenkalkül. *Comptes rendus des séances de la Société des sciences et des lettres des Varsovie Classe III* (23): pp. 30-50.

[31] Gaines, B. R. 1976. Fuzzy reasoning and the logics of uncertainty. *Proceedings of IEEE 6th International Symposium on Multiple-Valued Logic*. IEEE Computer Society Press. pp. 179-188.

[32] Rescher, N. 1969. *Many-Valued Logic*. New York: McGraw-Hill.