THE CNN UNIVERSAL MACHINE IS AS UNIVERSAL AS A TURING MACHINE

by

Kenneth R. Crounse and Leon O. Chua

Memorandum No. UCB/ERL M95/29

5 March 1995

THE CNN UNIVERSAL MACHINE IS AS UNIVERSAL AS A TURING MACHINE

by

Kenneth R. Crounse and Leon O. Chua

Memorandum No. UCB/ERL M95/29

5 March 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

The CNN Universal Machine is as Universal as a Turing Machine

Kenneth R. Crounse * Leon O. Chua

Abstract

It is shown that the simplest integrated circuit implementations of the CNN Universal Machine can play the 'game of life', and are therefore equivalent to a Turing Machine. In addition, a constructive proof is given for the direct implementation of general first-order cellular automata on such machines.

1 Introduction

The CNN with certain capabilities has been shown to be able to run the famous 'Game of Life' cellular automaton. The game of life played on a large enough array is known to be a universal computer [1]. Therefore, these CNNs are universal as well [2]. This approach was presented prior to the introduction of the machine concept for CNN arrays. Therefore, the methods require either multiple layers, complex template nonlinearities, or discrete time operation.

It is unlikely that such complicating elements would be included in the first CNN chip designs. It is therefore an important question to determine the simplest CNN Universal Machine implementation which retains universality in the sense of Turing. Here we show that the simplest CNN Universal Machines can implement the game of life – with only single layer continuous time CNN dynamics, linear templates, local logic memory, and a local logic unit, and are therefore, indeed, universal.

Next, we show a general constructive proof that the same architecture can directly implement arbitrary first-order cellular automata. Past approaches have used either discrete time operation and complex nonlinearities [3], multiple layers [4, 5], or time varying templates [6]. By using the CNN Universal Machine architecture, we show that these complications are unnecessary.

2 Background

2.1 The CNN Universal Machine

The CNN Universal Machine was first introduced in [7] to enable the power of the original Cellular Neural Network [8, 9] to be fully exploited. This advance embeds the standard CNN into a machine which can control the CNN parameters and manipulate and multiplex its inputs and outputs in a programmable manner. Current chip designs [10] exceed the capabilities of the machine shown in Figure 1, which is the 'minimal' architecture used for this discussion. High level programming languages are being developed which can be compiled into CNN templates and control sequences.

*Sponsored under the Joint Services Electronics Program, Contract Number F49620-94-C-0038. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

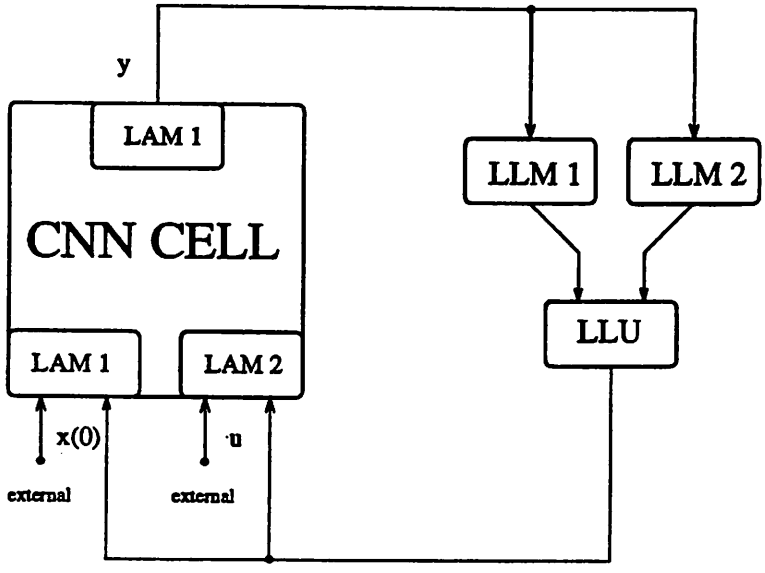

Figure 1: A simplified CNN Universal Machine register transfer diagram, with the obvious implicit multiplexing. The main features are the CNN cell where the analog transient occurs, the Local Analog Memories (LAM) which hold state and input, the Local Logic Memories (LLM) which can each store a binary value for each cell, and the Local Logic Unit (LLU) which can perform arbitrary intra-cell logic functions on the contents of the LLM.

2.2 First-Order Cellular Automata

The game of life is a first-order cellular automata, so we first discuss this class of systems.

Cellular automata are dynamical systems which are discrete in space, state, and time. We will speak of an array of cells. Each cell (i, j) holds a state $s_{i,j}(n)$ at time n. For a first-order CA, the state can only take on one of two values and the state values evolve in time according to a state transition rule which gives the next state as a function of the current states of a cell and its immediate 8-neighbors only. For our purposes, we assume that the transition rule is space invariant, that is, it is the same for every cell. Since the state transition rule can be an arbitrary boolean function of the eight neighbor's states and a cell's own state, there are an enormous $2^{2^9} > 10^{154}$ possible systems in this class, although many are symmetric.

In the following discussion we will use $+1$ to mean 'on', 'alive', 'logical TRUE', and -1 to mean 'off', 'dead', or 'logical FALSE'.

The game of life is discussed extensively elsewhere. It is called a 'totalistic' CA since the state transition rule depends only on the sum of alive neighbors. Following the approach in [2], it is convenient to describe the state transition rule in the following manner: " a cell will be alive (in the next generation) if and only if at least 3 of the 9 cells in its 3×3 neighborhood are alive and at most 3 of its 8 neighbors are alive."

3 Implementation of Life

The above formulation of the Life rule is important for a straightforward implementation on a CNN because it is in terms of the simple ANDing of two linearly separable boolean functions. It is well known that such functions can be expressed in terms of a simple linear threshold layer, which can be implemented on a CNN by a B-template and a thresholding A-template [4].

Number of 9-neighbors alive (+1)	Number of 9-neighbors dead (−1)	$\sum_{k,l=-1}^{1} s_{i+k,j+l} + 4$	$L_{i,j}$
0	9	-5	-1
1	8	-3	-1
2	7	-1	-1
3	6	+1	+1
4	5	+3	+1
⋮	⋮	⋮	⋮
Number of 8-neighbors alive (+1)	**Number of 8-neighbors dead (−1)**	$\sum_{k,l=-1\neq(0,0)}^{1} -s_{i+k,j+l} - 1$	$M_{i,j}$
0	8	+7	+1
1	7	+5	+1
2	6	+3	+1
3	5	+1	+1
4	4	-1	-1
⋮	⋮	⋮	⋮

Table 1: A demonstration that the boolean functions L and M can be written in the form of linear threshold functions.

For a given cell (i, j), define $L_{i,j}(n)$ to be true if at least 3 of the 9 cells are alive at time n and $M_{i,j}(n)$ to be true if at most 3 of its 8 neighbors are alive at time n. Then the Life rule can be written

$$s_{i,j}(n+1) = L_{i,j}(n) \text{ AND } M_{i,j}(n)$$

Now, for CNN implementation, we can write these boolean functions in a linear threshold form as follows:

$$L_{i,j} = \text{sgn}\left(\sum_{k,l=-1}^{1} s_{i+k,j+l} + 4\right)$$

$$M_{i,j} = \text{sgn}\left(\sum_{\substack{k,l=-1 \\ k,l\neq(0,0)}}^{1} -s_{i+k,j+l} - 1\right)$$

which can be verified by simply writing out the possibilities for the number of 'alive' neighbors, as shown in Table 1.

From this point, the mapping on to a CNN Universal Machine is straightforward. The two separable functions of the neighborhood can be formed by letting the current state of the CA be loaded into the input u and using the B-template to perform a weighted sum. The bias term I is used to determine the value which this sum is thresholded about. The A-template cancels the state self-feedback in the linear region and provides stability in the saturation region, and so the initial condition can be anything satisfying $|x_{i,j}(0)| < 1$. Specifically, the functions $L_{i,j}$ and $M_{i,j}$ can be implemented by the following templates:

$$\mathbf{A}_L = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B}_L = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \mathbf{I}_L = +4$$

$$\mathbf{A}_M = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B}_M = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 0 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \mathbf{I}_M = -1$$

The ANDing of the results can be performed by the LLU, or by another CNN template. See Figure 2 to see a flow chart of the complete algorithm when the LLU is used to perform the conjunction.

4 Implementation of general First Order Cellular Automata

There has been a lot of past interest in building circuits and accelerators [11, 12] to perform cellular automata operations for image processing [13], random number generation, and simulation of physical systems, to name a few. Therefore, even though the CNN Universal Machine was just shown to be able to perform anything a digital computer can, it is interesting to show that general cellular automata can be implemented in a direct manner through CNNUM algorithms.

A first-order cellular automata can be written in the following form:

$$s_{i,j}(n+1) = f\left[s_{k,l}(n)\right]$$

where $f[\cdots]$ is a boolean function of the nine variables $s_{k,l}(n)$ in the neighborhood of (i,j). Such a boolean function can always be written as a truth table where every possible neighborhood configuration is listed along with its corresponding next state. Each of these neighborhood configurations has a corresponding boolean expression, called a minterm, which is TRUE if and only if the neighborhood is in that configuration. Then, the function $f[\cdots]$ can always be written as the sum (OR) of the minterms for which the next state is to be TRUE. This is equivalent to explicitly checking for every neighborhood configuration for which the cell should be 'on' in the next step.

As an example, consider a state transition function which indicates alive when the cell is on a vertical, horizontal, or 45° line of alive cells. This can be expanded to checking for the four neighborhood cases shown in Table 2. The corresponding minterms are also given.

The complement of minterms are called *maxterms*. By a similar argument a boolean function can always be written as the product (AND) of maxterms, each of which corresponds to a 'next state' entry of FALSE in the truth table. This is equivalent to explicitly checking for every neighborhood configuration for which the cell should be 'off' in the next step. Since for a function of nine variables the truth table has 512 entries, it can always be written either as a sum (OR) of no more than 256 minterms or the product (AND) of no more than 256 maxterms.

It is simple to show that minterms and maxterms are both linearly separable and can be implemented by a linear threshold class CNN. Consider minterm 1 from Table 2. For each variable which needs to be TRUE in order for the minterm to be TRUE make the corresponding weight $+1$. For variables which are to be FALSE the corresponding weight is made to be -1. Then, it is clear that the weighted sum will be maximal and equal to 9 when the logical conditions of the case are met. If just one variable dissents the sum will be only 7 and will decrease further as more variables do not agree with the case we are checking. Then, to make the case selection,

$$\underset{\text{Case 1}}{\begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix}} \quad \underset{\text{Case 2}}{\begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}} \quad \underset{\text{Case 3}}{\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}} \quad \underset{\text{Case 4}}{\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}}$$

$$\begin{aligned}
\text{Minterm 1} &= (\bar{s}_{-1,-1})(\bar{s}_{-1,0})(\bar{s}_{-1,1})(s_{0,-1})(s_{0,0})(s_{0,1})(\bar{s}_{1,-1})(\bar{s}_{1,0})(\bar{s}_{1,1}) \\
\text{Minterm 2} &= (\bar{s}_{-1,-1})(s_{-1,0})(\bar{s}_{-1,1})(\bar{s}_{0,-1})(s_{0,0})(\bar{s}_{0,1})(\bar{s}_{1,-1})(s_{1,0})(\bar{s}_{1,1}) \\
\text{Minterm 3} &= (s_{-1,-1})(\bar{s}_{-1,0})(\bar{s}_{-1,1})(\bar{s}_{0,-1})(s_{0,0})(\bar{s}_{0,1})(\bar{s}_{1,-1})(\bar{s}_{1,0})(s_{1,1}) \\
\text{Minterm 4} &= (\bar{s}_{-1,-1})(\bar{s}_{-1,0})(s_{-1,1})(\bar{s}_{0,-1})(s_{0,0})(\bar{s}_{0,1})(s_{1,-1})(\bar{s}_{1,0})(\bar{s}_{1,1})
\end{aligned}$$

Table 2: The minterm will be TRUE if and only if the neighborhood is in the corresponding case configuration.

we should threshold around 8, which is equivalent to adding -8 and thresholding around zero. The associated CNN templates to check for case 1 are given by:

$$\mathbf{A}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B}_1 = \begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} \qquad \mathbf{I}_1 = -8$$

This constructive method can be applied to produce any of the 512 minterms. Since each maxterm is just the complement of a minterm, they can be formed by inverting both the weights in the B-template and the bias.

The OR or AND terms can be implemented by the LLU or through the CNN dynamics. See Figure 3 for a flowchart of a CNNUM algorithm using the LLU to implement the ORing of four minterms, for example those found in Table 2. Note that although the method may consume hundreds of CNNUM steps for every CA iteration (for instance, the game of life would require 84 templates), each happens very quickly and the template programming can be straightforwardly automated from any description (i.e., truth table, boolean function) of the state transition function. Also, with more LLMs, it is possible to simulate higher order CAs (i.e. more states per cell) by this method.

References

[1] E. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your mathematical plays*, vol. 2, ch. 25, pp. 817–850. New York: Academic Press, 1982.

[2] L. O. Chua, T. Roska, and P. L. Venetianer, "The CNN is as universal as the Turing Machine," *IEEE Transactions on Circuits and Systems-I*, vol. 40, pp. 289–291, 1993.

[3] P. L. Venetianer, P. Szolgay, K. R. Crounse, T. Roska, and L. O. Chua, "Analog combinatorics and cellular automata – key algorithms and layout design," Report DNS-7-1994, Analogical and Neural Computing Laboratory, Computer and Automation Institute, Hungarian Academy of Sciences, Nov. 1994.

[4] L. O. Chua and B. E. Shi, "Exploiting Cellular Automata in the design of Cellular Neural Networks for binary image processing," Memorandum UCB/ERL M89/130, University of California at Berkeley Electronics Research Laboratory, Nov. 1989.

[5] L. O. Chua and B. E. Shi, "Multiple layer Cellular Neural Networks - a tutorial," in *Algorithms and Parallel VLSI Architectures* (F. Deprettere and A. V. der Veen, eds.), pp. 137–168, Elsevier Science Publishers, 1991.

[6] Z. Galias, "Designing Cellular Neural Networks for the evaluation of local boolean functions," *IEEE Transactions on Circuits and Systems-II*, vol. 40, pp. 219–223, Mar. 1993.

[7] L. O. Chua and T. Roska, "The CNN Universal Machine, part 1: The architecture," in *Second IEEE International Workshop on Cellular Neural Networks and Their Applications, Proceedings*, pp. 1–10, 1992.

[8] L. O. Chua and L. Yang, "Cellular Neural Networks: Theory," *IEEE Transactions on Circuits and Systems*, vol. 32, Oct. 1988.

[9] L. O. Chua and L. Yang, "Cellular Neural Networks: Applications," *IEEE Transactions on Circuits and Systems*, vol. 32, Oct. 1988.

[10] J. M. Cruz, L. O. Chua, and T. Roska, "A fast, complex and efficient test implementation of the CNN Universal Machine," in *Proceedings of the Third IEEE International Workshop on Cellular Neural Networks and Their Applications*, pp. 61–66, Dec. 1994.

[11] T. Toffoli, *Cellular Automata Machines.* Cambridge, MA: The MIT Press, 1987.

[12] A. P. Marriott, P. Tsalides, and P. J. Hicks, "VLSI implementation of smart imaging system using two-dimensional cellular automata," *IEE Proceedings G (Circuits, Devices and Systems)*, vol. 138, pp. 582–586, Oct. 1991.

[13] K. Preston Jr. and M. J. B. Duff, *Modern Cellular Automata: Theory and Applications.* New York: Plenum, 1984.

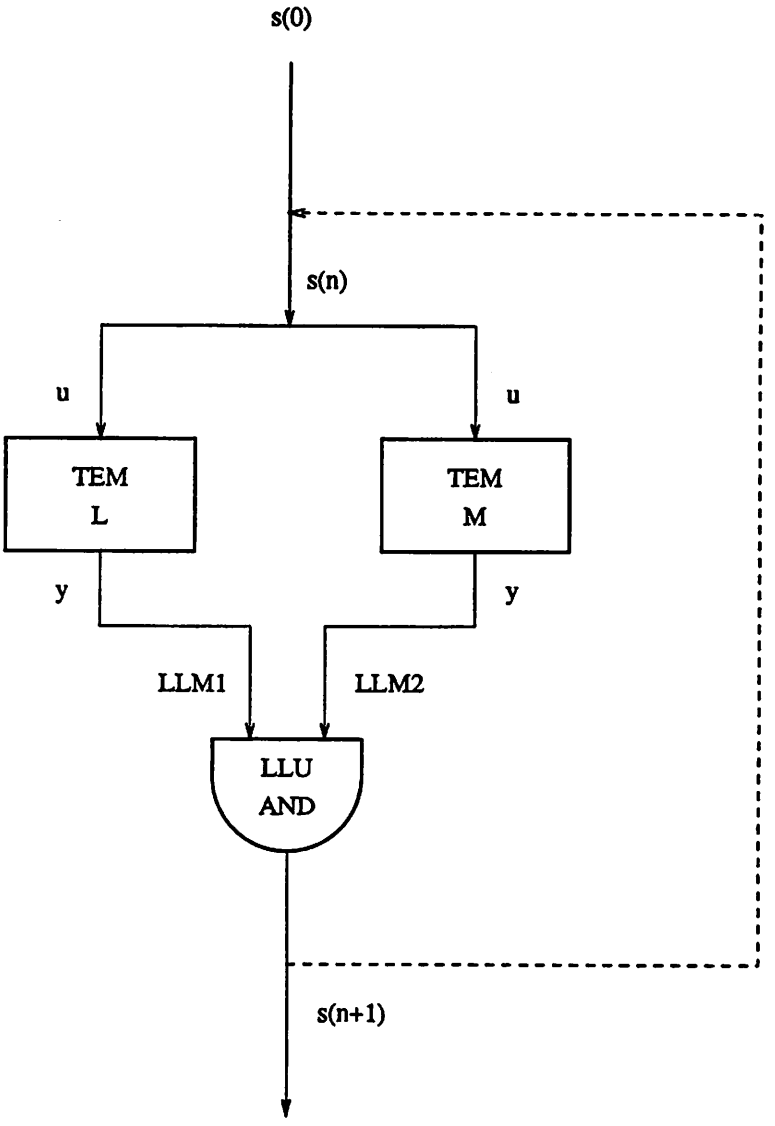

Figure 2: A flow chart of the CNN Universal Machine Algorithm for implementing the game of Life.

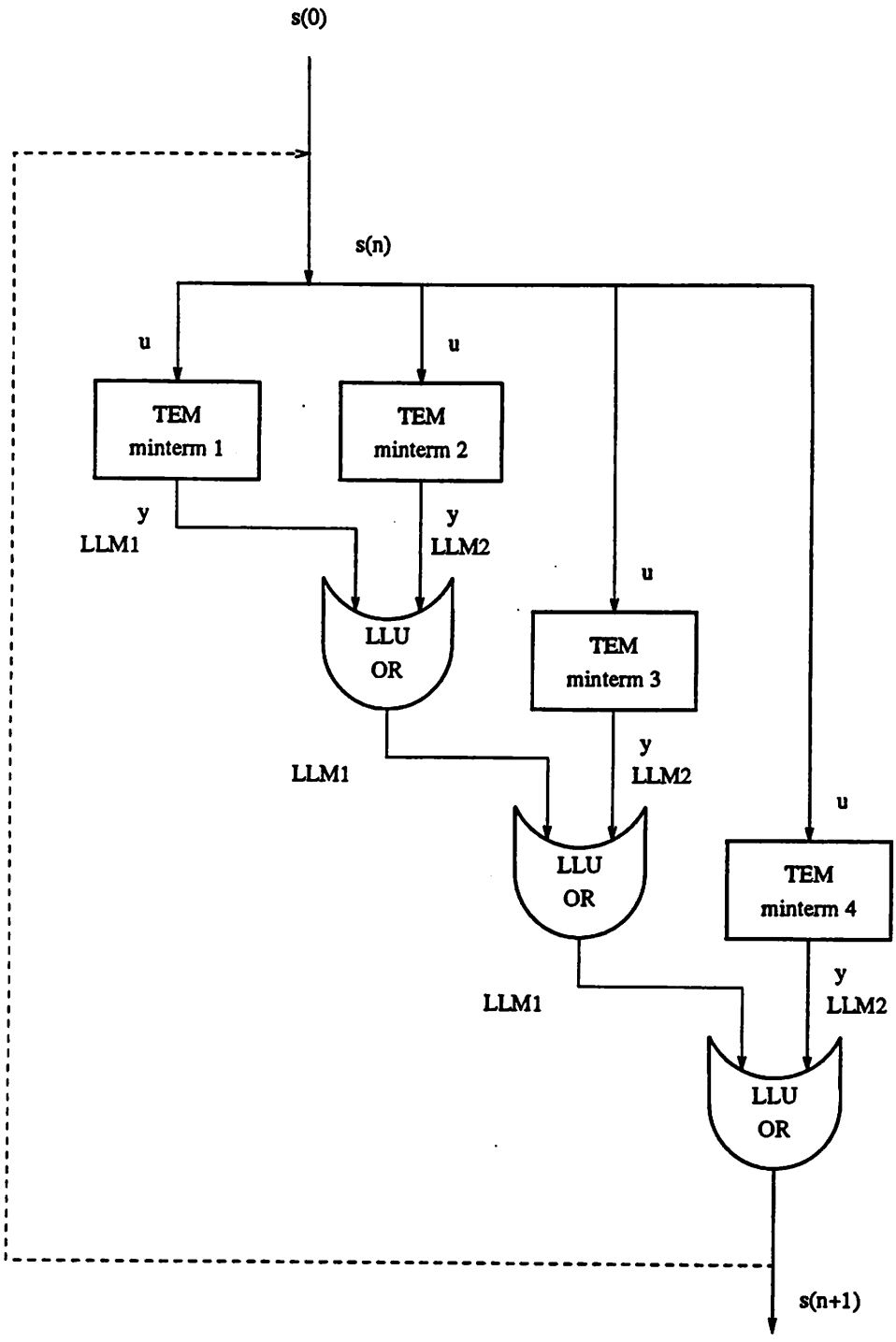

Figure 3: A flow chart of the CNN Universal Machine Algorithm for implementing a cellular automata with a 4-minterm state transition function. The structure can be extended indefinitely. In general the TEM blocks can be any linearly separable boolean operation.