Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

Implementation of Combinational Logic Circuits Using Nearest-Neighbor One-Dimensional Four-State Cellular Automata

A. Javan*[1], M. Jafarpour†[2], A. Moeini‡[3] and M. Shekaramiz§[4]

[1,2,3]University of Tehran, College of Engineering, Faculty of Engineerng Science, Department of Algorithms and Computation.

[2]Department of Electrical and Computer Engineering, Utah State University, Logan, UT 84322-4120, USA.

ABSTRACT

Cellular automata are simple mathematical idealizations of natural systems. They consist of a lattice of discrete identical sites, each site taking on a finite set of, say, integer values. Over the years, scientists have been trying to investigate the computational capabilities of cellular automata by limiting the dimension, neighborhood radius, and the number of states. In this article, we represent a novel implementation of combinational logic circuits using nearest-neighbor one-dimensional four-state cellular automata (CA). The novelty behind the proposed model is the reduction of the required number of states and yet being able to implement combinational logic-circuits in the conventional CA fashion. This can open a new window to the computation using cellular automata.

Keyword: Cellular Automata, Cellular Machine, Combinational Logic Circuits, Universality.

ARTICLE INFO

Article history:
Received 12, February 2020
Received in revised form 11, October 2020
Accepted 14 November 2020
Available online 30, December 2020
Research paper

*ajavan@ut.ac.ir
†Corresponding author: M. Jafarpour. Email: m.jafarpour@ut.ac.ir
‡moeini@ut.ac.ir
§mohammad.shekaramiz@aggiemail.usu.edu

AMS subject Classification: 68Q80.

1 Introduction

Cellular automata A is formally a tuple (d, S, N, f), where d is the dimension of space, S is a finite set of states, N a finite subset of Z^d is the neighborhood, and $f : S^N \to S$ is the local rule or transition function of the automaton. A configuration of a cellular automaton is a coloring of the space by S, which is an element of S^Z. The global rule $G : S^{Z^d} \to S^{Z^d}$ of a cellular automaton maps a configuration $c \in S^{Z^d}$ to the configuration $G(c)$ obtained from applying f uniformly in each cell: for all position $z \in Z^d, G(c)(z) = f(c(z + v_1), ..., c(z + v_k))$ where $N = \{v_1, , v_k\}$. A space-time diagram of a given cellular automaton is a mapping $\Delta \in S^{N \times Z^d}$ such that for all time step $t \in N$, $\Delta(t + 1) = G(\Delta(t))$ [24]. In other words, cellular automaton (CA) is discrete dynamic systems having a simple structure but with complex self-organizing behaviors [37]. Such complex behaviors are achieved by encoding computing devices of the universal class of machines.
Cellular automaton is considered to be used in computer processor implementations, cryptography, error correction, etc. [8, 19, 32]. Theoretically, any cellular automaton can be defined as discrete n-dimensional lattice of cells. Based on this definition, it can be imagined as one-dimensional, two-dimensional, , n-dimensional CA. The atomic components of the lattice can be differently shaped: for example, a 2D lattice can be composed of triangles, squares, or hexagons. Usually homogeneity is assumed. All cells are qualitatively identical [4].
As the simplest class, one-dimensional (1D) automaton is assumed as a sequence of cells which take on a finite set of possible values (states) in discrete time steps according to deterministic and predefined rules involving neighbor cell(s) [36].
Regardless of designing any class of cellular automaton, the main goal is to achieve universal automata.
According to Davis [11], a universal program Φ is a program that can compute the output based on the given inputs $x_1, x_2, ..., x_n$ and desired function $\Psi^{(n)}$ as follows:
$\Phi^{(n)}(x_1, x_2, ..., x_n, y) = \Psi^{(n)}_{\rho}(x_1, x_2, ..., x_n)$ where $\#(\rho) = y$.
In [10, 34], universality is the property described as the capability of performing different tasks with the same underlying construction by just programming them in a different way. Universal systems are effectively capable of emulating any other system. Although digital computers are universal, proving that idealized computational systems are universal can be extremely difficult and technical. Nonetheless, any system that can be translated into another system, which is known to be universal, must itself be universal. For example, a universal Turing machine (UTM) is a Turing machine that can simulate an arbitrary Turing machine on arbitrary input.
Specific universal Turing machines, universal cellular automata (in both one and two dimensions), and universal cyclic tag systems are known, although the smallest universal

example is known only in the case of elementary cellular automata [10, 34].
Any CA regardless of beig universal or not, leads to diversity of behaviors for which it is designed. Wolfram classified all CA behaviors in four groups [35]

1. The ones that spatially have homogenous state
2. The ones that have a sequence of simple stable or periodic structures
3. The ones that have chaotic aperiodic behavior
4. The ones with complicated localized structures, some propagating

As being the simplest type of complex machines, universal machines are the sum of all behaviors.The notion and construction of a universal cellular automaton are as old as the formal study of the object itself, starting with the work of von Neumann [25] on self-reproduction in the 1940s, and using cellular automata under suggestions by Ulam [33]. Originally, universality consisted of cells with four orthogonal neighbors and 29 possible states as proposed by von Neumann [33]. Throughout the years, however, it was shown that cells consisting of fewer neighbors and less number of possible states would also be capable of simulating a universal Turing machine. Ollinger reviews such evolution over time, in detail [25].
Further works based on the idea of von Neumann leads to various improvements and discussions on the encoding of Boolean circuits, the different organs that compose the machine and the transmission of signals. Some of these works are included in [1, 5–7, 18, 24, 30, 31].
As it is mentioned before, construction of a universal CA started with the work of Neumann [12, 26, 28, 33] and continued by others including Codd [9]. Following the principle of von Neumanns idea on self-reproduction, Codd was able to reduce drastically the complexity of the automaton. Codd's two-dimensional rule used eight states with the von Neumann neighborhood. Signals were conveyed by pairs of states (an oriented particle) moving between walls and reacting upon collision. This cellular automaton was also universal for Boolean circuits and so intrinsically universal [9, 25].
Following the work of Turing, a Turing-universal cellular automaton is an automaton encompassing computational power of the class of Turing machines, completely. Smith III simulated such an automaton using CA with the first neighbors and 18 states [29]. However, Turing-universality is not the only reasonable kind of universality which one might expect from cellular automata [33]. More literally, a machine is computationally universal if it is able to compute any computable function (indicated as a part of the entry). This corresponds also to the common approach of the computer, the hardware is universal and the program to be executed is stored in main memory (like the data to process), and is part of the input as far as the hardware/operating system is concerned [14].
The Game of Life introduced by Conway [3, 15] is certainly among the most famous cellular automata and the first rule which is proved that is universal by analysis of a given rule rather than on purpose construction. A modern exposition of the Game of Life universality and a proof of its intrinsic universality are presented by Durand [13].

In [22], it was shown that intrinsic universality of one-dimensional Cellular Automata for the nearest neighbor is possible by only four states, with a similar proving approach to [10].
Recent signs of progress are reviewed in the following researches. In [16], the construction of small and simple two-dimensional reversible cellular automata was improved. Ollinger used simulation techniques between cellular automata and strong intrinsically universal cellular automata, showing that it can be constructed with few states (i.e. 6 states) [23]. Cook [10] and Wolfram [34] showed that very small universal cellular automata cannot be constructed, but rather they have to be obtained by analysis. Cook was able to prove the Turing-universality of the two-states first-neighbors so-called rule 110 by analyzing signals generated by the rule and their collisions.
In 2006, Neary and Woods proved the prediction problem of the rule 110 being P-complete via careful analysis and modification of Turing machines simulation techniques by tag systems [20]. Banks constructs a family of very small cellular automata (two-dimensional, von Neumann neighborhood, very symmetric, four to two states) simulating Boolean circuits in a very simple and modern way (signals moving in wires, Boolean gates on collisions). He identified and explicitly used the property of intrinsic universality and gave a transformation to construct relatively small universal one-dimensional cellular automata with large neighborhoods starting from two-dimensional ones (re-encoding it into a one-dimensional first-neighbors automaton with 18 states) [33]. Construction of a two-dimensional four state universal cellular automaton in the spirit of Banks is provided by Noural and Kashef [21].
The limits of constructed small intrinsically universal cellular automata are converging towards analyzed cellular automata. Using particles and collisions, the authors in [10, 34] were able to construct a four-state first-neighbors one-dimensional intrinsically universal cellular automaton. It is shown in [34] that "AND" and "NOT" gates can be implemented by 1D nearest neighbor CA using five states.
In this paper, we offer an improvement to the five-state 1D nearest neighbor CA [34](pages 667-668) by presenting an algorithm that relies on less number of states (four states) for such cellular automata. In addition, we show that the presented model will be universal in terms of combinational Boolean circuits. However, the Turing universality of the proposed CA is an issue that remains open for future studies. in order to take steps towards Turing universality, the CA should simulate all the bahaviours of the Turing machine including the memory. The digital circuits fall into two categories of combinational and sequential. The combinational Boolean circuits are referred as time-independent logic [27], in which the output is a pure function of the present input only. While for the sequential circuits, the output depends on the history of the input, as well. In other words, the sequential logic has memory, while the combinational logic has not.
In order to achieve this, we use NAND and NOR gates, because they are universal gates. This means that they can simulate any other gates [17]. This paper is organized as follows. We first present the Definition of the Problem in Section 2. CA rules are covered in Section 3. Then we will discuss the proposed cellular construction algorithm in Section 4. In Section 5, analysis of the algorithm is covered. Section 6 discusses halting. Section

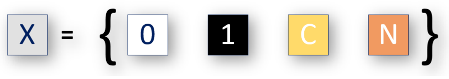

Figure 1: Four States of the proposed CA

7, will cover the reduction of the length of the expression. Cell (memory) and time complexities will be discussed in Section 8, and Section 9 illustrates future work. Finally, the conclusion is covered in Section 10.

2 Problem Definition

The ideal end to this paper is to construct a computational machine or a computational decision making machine. The ideal automaton is achieved only when the behavior is universal. As it is discussed in the previous section, the computational constructions could be considered universal in several aspects including Turing universality and any other types of universality such as the one that this paper is following. This paper is aimed to focus on implementation of a cellular automata which is universal in the scope of combinational logic circuits. This means that, the proposed CA can implement any type of combinational logic circuits. Let the proposed CA be named as CLA (i.e. Cellular Logic Automata). This paper does not claim to propose a Turing-universal CA but claims that the proposed model (i.e. CLA) can take an effective step towards universality in terms of logic circuits in the future.

3 CA Rules

According to formal definition, The CA rules as one of main components of CAs, are used to determine the state of a cell in the next generation based on its own and neighbors previous states. The CA rules can be categorized into four main groups. These four groups are commutative, NAND, conversion, and reduction rules. The state denoted by X represents ”dont-care”, (this notation is considered in formal notations of logic circuits design [17]) which can be replaced by any of the four possible states (”C” stands for Commutation/Carry, and ”N” stands for NAND). Since this is the first place that it was shown, we need to talk about it here.

3.1 Moving Forward Rules

These rules are used to move forward the 0 or 1 values. In order for a forward rule to be activated, a ”C”-containing cell has to be placed to the right of a (0/1)-containing cell. In this case, the values of the two cells are exchanged. This Rule is illustrated in Figure 2. In the first image from left, the middle cell contains 0 and its nearest neighbor on the right hand contains ”C”. Thus the values of these two cells will be exchanged and the value will go forward eventually. In Figure 2 the case is shown for other possibilities too.

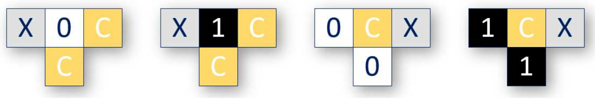

Figure 2: The Moving Forward Rules

3.2 NAND Rules

The NAND rules are applied when two (0/1)-containing cells are placed adjacent to one another, regardless of the values of other cells. It is noteworthy that this group of rules is the only group of rules that need to be changed in case of a NOR-equivalent implementation. Figure 3 shows the NAND operation. For example, if two neighbor cells contain 0, the left cell in the next step will take the state 1 ($(0 \wedge 0)' = 1$).

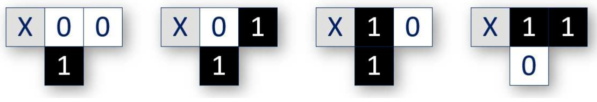

Figure 3: The NAND Rules

3.3 Conversion Rules

The NAND operation is executed through conversion rules. The NAND operation does not require taking into account the status of the state of the front cell, since the (0/1)-containing cells are adjacent to each other only when the state of the next cell is N. Figure 4 shows the conversion rules and how they work.

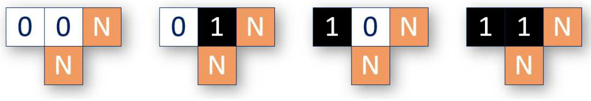

Figure 4: The Conversion Rules

3.4 Reduction Rules

As mentioned earlier, a NAND operation is activated only when two adjacent cells contain 0/1 values and the nearest neighbor to the right cell contains N.the execution of NAND rules might lead to N-containing cells are adjacent to each other. Therefore, the states of such cells will have to be converted to ”C”. This rule is illustrated in Figure 5.

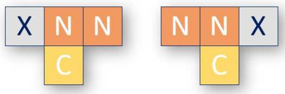

Figure 5: The Reduction Rules

4 The Proposed Cellular Construction Algorithm

It is straightforward to show that NAND and NOR gates are universal gates. This is essentially due to the fact that we can use either of them to implement all digital circuits. Without loss of generality, here we use the NAND gate to represent Boolean functions. In this section, we present our proposed Cellular Construction Algorithm in five steps which will make the logic circuit work. This algorithm forms the logic circuit and shows how the inputs must be ready for the process. The central logic to this five-step cellular construction algorithm is to use various cell combinations to indicate the current state of the system. In other words, the cells are controlled such that each state would represent a specific function.The steps of the algorithm are shown in algorithm 1 .

Algorithm 1 Cellular Construction Algorithm

```
    Input: Logic Circuit's Boolean Expression $f(x_1, x_2...x_n)$
    Input: $XV = \{x_i\ Values, \forall i \in 1 \ldots n\}$
    Output: Corresponding Cellular Automatas Cells

1: TINf ← CovertToTwoInputNAND(f)    ▷ Representation of the Boolean expression in terms of NAND gates. In this step, all AND, OR, and NOT gates of the Boolean function are converted to two-input NAND gates.
2: POTINf ← PostOrder(TINf)    ▷ Post-order expression. The resulting function from step 1 is expressed in terms of variables and NAND symbol (').
3: Cf ← CovertToCells(POTINf)    ▷ Expression formatting. At this stage, three carrier states "CCC" are added before every variable, while one carrier state "C" is added before every NAND symbol ('). We also replace every NAND operator with the state "N". In addition, two "CN" cells will be added to the end of the CA cells.
4: CA ← ReplaceValues(Cf,XV)    ▷ Variable initialization. At this stage, all variables are initialized to either 0 or 1 based on the input values.
5: return CA    ▷ The string resulting from the above four-step algorithm would always consist of a maximum of four different states: States "0" and "1" for representing the bits, one state "N" for representing the NAND operand, and state "C" for representing the carrier symbol. Each character of the resulting string is then placed into an individual cell. Finally, by applying the rules given in section 2, the result of the Boolean expression (Logic circuit) will appear in the second last cell following by a "N" cell, while the other cells will contain the state "C".
```

In order to have a better understanding of the proposed CA, here we present the modeling of a digital circuit with an example. Consider the Boolean function presented in equation 1.

$$f(x, y, z) = (x \wedge y') \vee (x \wedge z) \tag{1}$$

The implementation of the equation 1 by two-input NAND gates, will be as equation 2.

$$CovertToTwoInputNAND(f) = ((x \wedge (y \wedge 1)')' \wedge (x \wedge z)')' \tag{2}$$

The post-order representation of the function is achieved by removing the AND symbols and parentheses, as seen in equation 3.

$$PostOrder(CovertToTwoInputNAND(f)) = xy1''xz'' \tag{3}$$

Converting the post-order template to cell states shown in equation 4.

$$C(f) = CCCxCCCyCCC1CNCNCCCxCCCzCNCNCN \tag{4}$$

Replacing variables by arbitrary values of $XV = \{x_{value}, y_{value}, z_{value}\} = \{1, 0, 0\}$ gives us the equation 5.

$$CA(C(f), XV) = CCC1CCC0CCC1CNCNCCC1CCC0CNCNCN \tag{5}$$

Figure 6 illustrates the starting state of CA, according to 5. The figure 7 shows the next state of CA. Therefore, the resulting CA that corresponds to the given function will be as Figure 8.

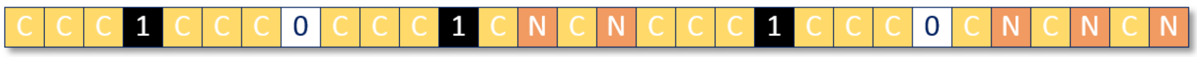

Figure 6: The CA Cells Corresponding to Function Mentioned in Equation 1.

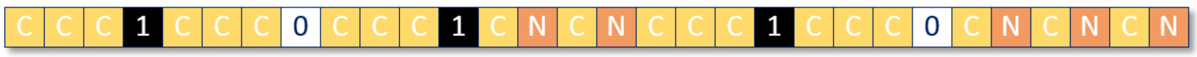

Figure 7: The Second state of CA Cells Corresponding to Function Mentioned in Equation 1.

The application of the rules described in Section 2 on cell states in each step, yields to solving the value of the function. The chronological orderings of cell-states for each step is given in Figure 8.

As shown in Figure 8, the final value of the given Boolean function will then be stored in the second last (second right-most) cell, in the halting state of the CA. The CA proposed in this paper is following the concept of universality proposed by Davis in Section 1. it is assumed that the proposed CA can compute any combinational logic circuit function with determined inputs and functions.

5 Analysis

The arrangement of two consecutive cells is exploited in this approach. For example, when two consecutive cells contain values (or states) of 0 and 1, one can conclude that the first bit has reached the "N" state and has remained in that position so that the second bit is beside it. This arrangement indicates that a cell with the state "N" is in front of the two bits; which in turn means that the NAND operation has to be performed on the aforementioned bits. Hence, we can predict the state of the next cellwithout observing its actual value. In fact, the NAND rules have been determined by using the same logic. The reason for placing two carrier states "CCC" before every variable is the avoidance of two variables reaching each other before they reach the NAND operator. Each NAND operation is done in two phase of conversion and reduction, explaining the placement of three carrier states "CCC" before every variable. Conversion rules lead to the placement of the two bits beside the N cell that indicates the end of the NAND operation. Therefore, it is necessary to place one carrier state "C" before every "N" to avoid the placement of two consecutive "N" states in the beginning. In the reduction rules, the idea that two numbers and an "N" state would mean the end of the operation is used to eliminate the NAND operands by replacing the "N" state by the "C" state.

6 Halting

As discussed earlier in section 4, once the operation is done, the result of the desired Boolean function will appear in one of the cells and move to the next cell periodically while all other cells contain the carrier symbol. This process can be enhanced by adding halting to the procedure in the following way: Two additional cells (C and N) are appended to the end (right side) of the CA. Upon arrival of the result to these cells, the value of the cell containing the result shifts one step to the right side and stops rotating. Therefore, the second last cell right before the cell containing N always contains the result of the Boolean expression.

7 Reduction of The Length of the Expression

The first step of the CA algorithm converts all gates to their equivalent NAND representation. In order to further improve the procedure, inversion rules can indeed be replaced by a NAND gate feeding the expression and 1, since X'=(X1)'. This technique can significantly reduce the length of the expression since the NAND representation is expected to have terms that are inverse of one another.

8 Cell and Time Complexity

We can compute the number of cells required for implementing the function. This is explained in the following example. For the case (xy)', 4 cells are needed for x, 4 for y, 2 for CN, and 2 for the last CN. Thus, the total number of cells is then

$$4 \times (Number\, of\, Inputs) + 2 \times (Number\, of\, NAND\, Gates) + 2. \tag{6}$$

In order to obtain the results of the logic circuit in the aforementioned cellular automata, the time complexity of the logic computations depends on the number of NAND operations being involved in the leftmost 0/1 bit.

9 Future Works

The modeling of logic circuits using CA is proposed in this paper. However, the minimum required CA for implementation of logic circuits on the proposed machine is unknown. The Boolean expressions to be processed are the ones that can be modeled using combinational universal logic circuits. Thus this model corresponds to the combinational universal logic circuits.
In Figure 9, the left, middle, and right blocks represent memory cells, logic circuit (Boolean expression) cells, and cells constraining final result, respectively. In the previous sections, the simulation of combinational gates is provided. Hereby, we call this machine as the Cellular Machine. The main question is that how many states are required to model this Cellular Machine. In order to implement the above machine, the rules have to be organized in such a way that provided for the Read/Write commands. Additionally, the rules must implement guiding the result of each block to the next block in the right as well as performing Read/Write executions on the left memory cell based on the given memory address. This CA machine cannot model sequential logic circuits. Thus, it is not possible to model some circuits such as Flip-Flop, etc. which prevents the capability of creating memory cells. In this model, the memory cells should be apart from logic circuits and thus there is a need to provide some cells as memory cells on the left side of the logic circuits part in this machine. If we decide not to use the Flip-Flop to model memory cells, the minimum number of states and the rules to build up this model still remains an open problem. The other problem is as follows. In order to build up such a Cellular Machine with specific states, what sort of physical structures such as atoms, molecules, etc. could be used so that they would cover the rules and states of this Cellular Machine?

10 Conclusion

In this work, we showed that the approach presented by Wolfram in [34] scould indeed be further optimized by eliminating a redundant state. The presented CA rules and cellular construction algorithm enable us to construct a four-state 1D nearest neighbor CA rather than a five-state, which is significant, compared to our proposed approach. A similar approach may be possible by deploying NOR gates instead of NAND gates. Therefore, CA rules satisfying the four-state requirements are to be designed.

References

[1] Arbib, MA., Simple self-reproducing universal automata, *Information and Control.* Apr 1966, 9(2):pp. #177-89#.

[2] Banks, ER., Universality in cellular automata, *In IEEE Conference Record of 1970 Eleventh Annual Symposium on Switching and Automata Theory, IEEE*, 1970 Oct, pp. #194-215#.

[3] Berlekamp, C., Conway, JH., Guy, RK., Winning Ways for your Mathematical Plays, Vol. 2, *ISBN: 978-1568811420*, 2003.

[4] Berto F, Tagliabue J., Cellular automata, 2012.

[5] Burks, AW., Essays on cellular automata, *University of Illinois Press*, 1970.

[6] Burks, AW., Toward a Theory of Automata Based on more Realistic Primitive Elements, textitInIFIP Congress, 1962, pp.
#379-385#.

[7] Burks, AW., Von Neumann's self-reproducing automata, *MICHIGAN UNIV ANN ARBOR LOGIC OF COMPUTERS GROUP*, Jun 1969.

[8] Chowdhury, DR., Basu, S., Gupta, IS., Chaudhuri, PP., Design of CAECC-cellular automata based error correcting code. IEEE Transactions on Computers. 1994 Jun;43(6):759-64.

[9] Codd, EF., Cellular Automata, *Academic Press, Inc.*, 1968.

[10] Cook, M., Universality in Elementary Cellular Automata. *Complex Systems*, 2004, 15(1): pp. #1-40#.

[11] Davis, M., Sigal, R., and Weyuker, EJ., Computability, complexity, and languages: fundamentals of theoretical computer science, Elsevier, 1994.

[12] Davis, MD., The Universal Computer: The Road from Leibniz to Turing, *W. W. Norton Co., Inc., New York, NY*, 2000.

[13] Durand, B., Rka, Z., The game of life: universality revisited, *Cellular automata, Springer, Dordrecht*, 1999, pp. #51-74#.

[14] Durand-Lose, J., Cellular automata, Universality of Computational Complexity: Theory, Techniques, and Applications, 2012, pp.#443-55#.

[15] Gardner, M., Mathematical games: The fantastic combinations of John Conways new solitaire game 'life', *Scientific American*, 1970 Oct, 223(4):pp. #120-3#.

[16] Imai, K., Morita, K., A computation-universal two-dimensional 8-state triangular reversible cellular automaton, *Theoretical Computer Science*, 2000 Jan, 231(2):pp. #181-91#.

[17] Mano, M., CR. Kime, CR., Logic and computer design fundamentals, *Upper Saddle River, NJ, Pearson Prentice Hall*, 2008.

[18] Moore, EF., Machine models of self-reproduction, *Proceedings of symposia in applied mathematics, American Mathematical Society New York*, 1962 Dec, Vol. 14, No. 1962, pp. #17-33#.

[19] Muhtaroglu, A., 4.1 Cellular Automaton Processor (CAP), Cellular Automaton Processor Based Systems for Genetic Sequence Comparison/Database Searching, Cornell University, 1996, pp. 6274.

[20] Neary, T., Woods, D., P-completeness of cellular automaton Rule 110, *International Colloquium on Automata, Languages, and Programming, Springer, Berlin, Heidelberg*, 2006 Jul, pp. #132-43#.

[21] Noural, F., Kashef, RS., A universal four-state cellular computer, *IEEE Transactions on Computers*, 1975 Aug, 100(8):pp. #766-76#

[22] Ollinger, N., Richard, G., Four states are enough!, *Theoretical Computer Science*, 2011, 412(12):pp. #22-32#.

[23] Ollinger, N., The quest for small universal cellular automata, textitInternational Colloquium on Automata, Languages, and Programming, Springer, Berlin, Heidelberg, 2002 Jul, pp. #318-29#.

[24] Ollinger, N., Universalities in cellular automata; a (short) survey, *Bruno Durand, JAC 2008, France, Regular paper track,* $\langle hal - 00274563\rangle$, 2008 Apr, pp. #102-18#

[25] Ollinger, N., Universalities in Cellular Automata, *Handbook of Natural Computing, G.*

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

[27] Savant, C.J., Roden, M., Carpenter, G., Electronic Design: Circuits and Systems, *ISBN: 0-8053-0285-9*, 1991, pp. 682.

[28] Shiffman, D., The Nature of Code: Simulating Natural Systems with Processing, *Self-published*, 2012.

[29] Smith III, AR., Simple computation-universal cellular spaces, *Journal of the ACM (JACM)*, 1971 Jul, 18(3):pp. #339-53#.

[30] Thatcher, JW., Self-describing Turing machines and self-reproducing cellular automata, *Essays on*, 1970.

[31] Thatcher, JW., Universality in the von Neumann cellular model, *Michigan univ. Ann arbor coll. of literature science and the arts*, 1964 Sep.

[32] Tomassini, M., Sipper, M., Perrenoud, M., On the generation of high-quality random numbers by two-dimensional cellular automata. IEEE Transactions on computers. 2000 Oct;49(10):1146-51.

[33] Von Neumann J., Burks, AW., Theory of self-reproducing automata, *Urbana: University of Illinois Press*, 1996.

[34] Wolfram, S., A New Kind of Science, *Champaign, Ilinois, US, United States: Wolfram Media Inc.* 2002.

[35] Wolfram, S., Cellular automata as models of complexity, *Nature*, 1984, 311: pp. #41924#.

[36] Wolfram, S., Computation Theory of Cellular Automata, *Comm. Math. Phys.*, 1984, 96: pp. #1557#.

[37] Wolfram, S., Universality and complexity in cellular automata, *Physica D: Nonlinear Phenomena*, 1984, 10(1-2): pp. #1-35#.

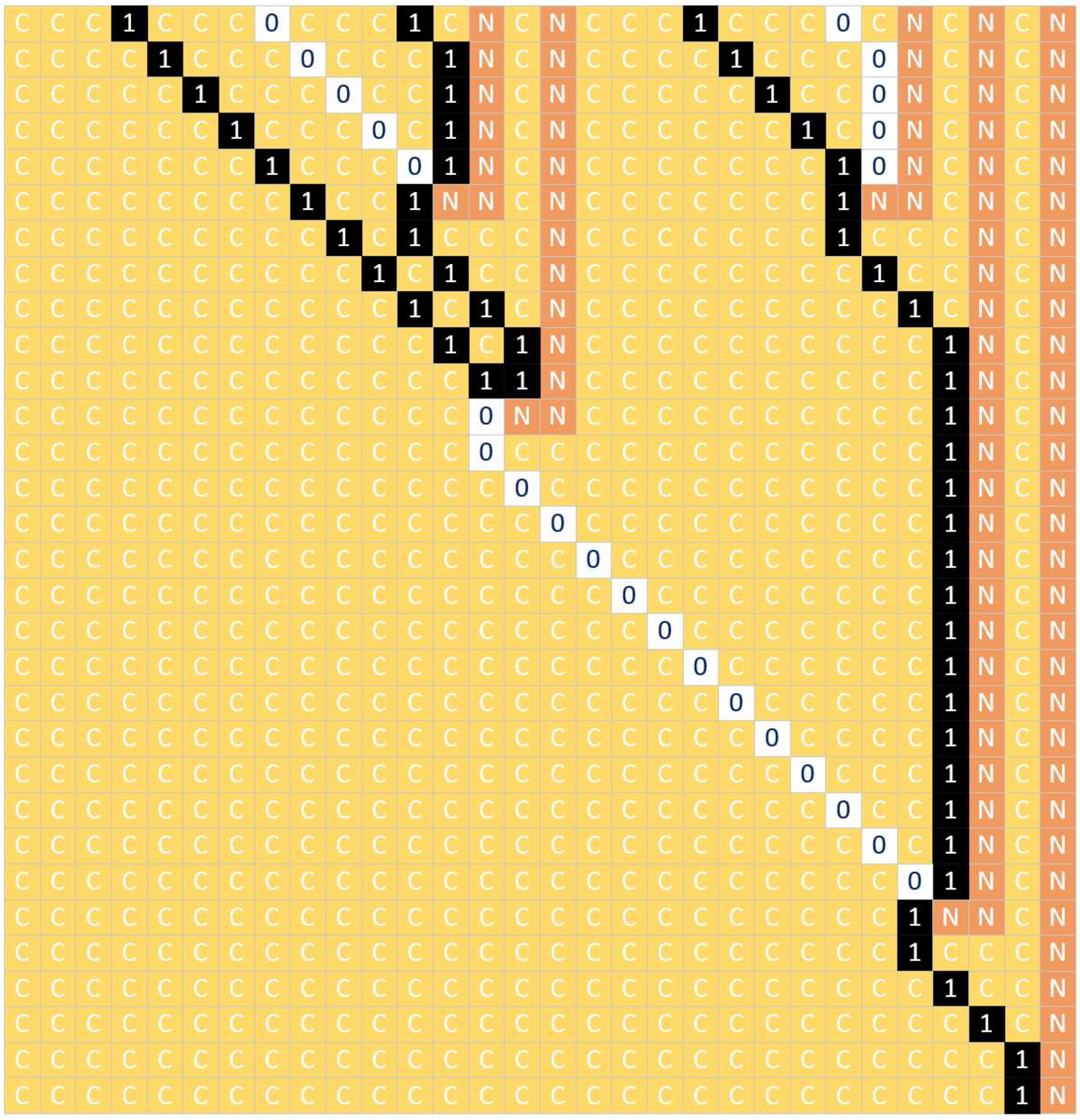

Figure 8: Time steps in the evolution of the CA corresponding to the function in Equation 1.

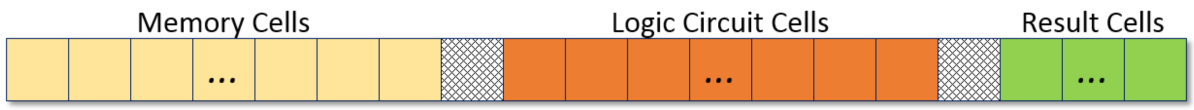

Figure 9: The Computational Logic Circuit Conceptual Model Using Cellular Automata (Cellular Machine).