# Reversibility in Asynchronous Cellular Automata

**Anindita Sarkar**[*]
**Anindita Mukherjee**[†]
**Sukanta Das**[‡]
*Department of Information Technology*
*Bengal Engineering and Science University, Shibpur*
*Howrah, West Bengal, India 711103*
[*]*anindita.sarkar10@gmail.com*
[†]*aninditait86@gmail.com*
[‡]*sukanta@it.becs.ac.in, Corresponding Author*

The reversibility issue of one-dimensional asynchronous cellular automata (ACAs) is addressed in this paper. The cells of ACAs are updated independently. The cellular automata (CAs) rules are classified as reversible and irreversible rules. The irreversible rules cannot configure reversible ACAs. The reversible rules may configure reversible ACAs depending upon the update of ACA cells. Finally, an algorithm is developed that outputs a sequence of ACA cells for a given CA rule to be updated to generate a cycle for a reversible ACA.

## 1. Introduction

Cellular automata (CAs), proposed in the early 1950s on a two-dimensional grid, are involved in a five-neighborhood interaction among the cells with 29 states per cell [1]. Later, the CA structure was simplified by a number of researchers and finally, a two-state three-neighborhood CA structure was proposed on a one-dimensional lattice [2]. This simplest version of CAs attracted a large number of researchers from various fields due to their simplicity and ability of modeling physical systems successfully. However, all such CAs are synchronous because all the cells of CAs update their states simultaneously.

The concept of asynchronous cellular automata (ACAs) was first developed on a one-dimensional lattice [3]. A formal definition of ACAs for the two-dimensional CA structure was provided in [4]. The one-dimensional ACAs were further studied in [5, 6]. S. Wolfram [6] refers to the ACAs as *sequential cellular automata*. The clocks of the ACA cells are independent, so the cells are updated independently.

The reversibility of synchronous CAs has been studied extensively for years [7–9]. However, reversibility of ACAs is an almost untouched issue. A very few papers on the issue of two-dimensional ACAs are found in the literature [10]. However, the reversibility of


https://doi.org/10.25088/ComplexSystems.21.1.71

one-dimensional ACAs is an unexplored field. In this scenario, we target exploring the issue for one-dimensional two-state three-neighborhood ACAs. We use the term *reversibility* in a classical sense—that is, starting from a CA state, a reversible ACA can reach to that particular CA state uniquely after a number of steps. During their evolution, unlike [3, 6], we consider that more than one ACA cell may be updated simultaneously. Based on the update of ACA cells in subsequent steps, we, as in [11], define an *update pattern* to know which cell is updated when. While an update pattern along with an initial state is given, the transition of CA states for an ACA can be observed. The update patterns play a major role in the reversibility of ACAs.

We have also identified a number of CA "rules" [2] as *irreversible rules*, which cannot configure reversible ACAs with any set of update patterns. Only *reversible rules* can configure reversible ACAs with a particular set of update patterns. An algorithm is also developed to find an update pattern of a cycle for some reversible ACA.

The paper is organized as follows. The preliminaries of CAs are provided in Section 2. Section 3 defines the reversibility of ACAs, and identifies the reversible and irreversible CA rules. The method to find an update pattern for a cycle of reversible ACAs is reported in Section 4. Section 5 concludes the paper.

## 2. Cellular Automata

CAs are the discrete spatially extended dynamical systems that have been studied extensively as models of physical systems. They evolve in discrete space and time. In their simplest form, as proposed by Wolfram [2], CAs consist of a lattice of cells, each of which stores a discrete variable at time $t$ that refers to the present state of the CA cell. The next state of a cell is affected by its present state and the present states of its neighbors at time $t$. In one-dimensional two-state three-neighborhood (self, left and right neighbors) CAs, the next state of each cell is determined as

$$S_i^{t+1} = f\left(S_{i-1}^t, S_i^t, S_{i+1}^t\right) \tag{1}$$

where $f$ is the next state function and $S_{i-1}^t$, $S_i^t$, and $S_{i+1}^t$ are the present states of the left neighbor, self, and right neighbor of the $i^{\text{th}}$ CA cell at time $t$. The function $f : \{0, 1\}^3 \mapsto \{0, 1\}$ can be expressed as a look-up table (see Table 1). The decimal equivalent of the eight next states (NS) is called a *rule* [2]. There are $2^8$ (256) CA rules in the two-state three-neighborhood dependency. Two such rules are 60 and 51 (Table 1). From the viewpoint of *switching theory*, a combination of the present states (PS in Table 1) can be viewed as the *min term* of a three-variable $(S_{i-1}^t, S_i^t, S_{i+1}^t)$ switching function. So, each column of the first row of Table 1 is referred to as rule min term (RMT).


https://doi.org/10.25088/ComplexSystems.21.1.71

| PS: | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 | Rule |
|---|---|---|---|---|---|---|---|---|---|
| (RMT) | (7) | (6) | (5) | (4) | (3) | (2) | (1) | (0) | |
| (i) NS: | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 60 |
| (ii) NS: | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 51 |

**Table 1.** Look-up table for rules 60 and 51.

The collection of states of all cells $(S_1^t, S_2^t, \ldots, S_n^t)$ at time $t$ is called a CA state on that time. If the leftmost and rightmost cells are the neighbors of each other (i.e., $S_0^t = S_n^t$ and $S_{n+1}^t = S_1^t$ for CAs with $n$ cells), the CAs are *periodic boundary* CAs. On the other hand, in *null boundary* CAs, $S_0^t = S_{n+1}^t = 0$ (null).

If all of the CA cells update their states simultaneously they are *synchronous* CAs. In *asynchronous* CAs, the cells are updated independently. Therefore, ACAs have decentralized control structure, and as a result, any number of ACA cells may be updated in a single time step. So, we consider, unlike [3, 6], that more than one—even all the ACA cells—may update their states simultaneously.

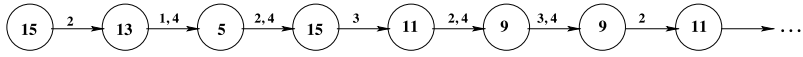


**Figure 1.** Partial state transition diagram of rule 60 ACA. The cells updated during state transition are noted over the arrows.

During their evolution with time, CAs (synchronous and asynchronous) generate a sequence of states. The next state of a CA can be determined in a synchronous CA configured with a particular rule. However, the next state of ACAs depends not only on the rule, but also on the cells that are updated at that time. We denote the set of cells, updated at time $t$, as $u_t$. Therefore, an *update pattern* $U = \langle u_1, u_2, \ldots, u_t, \ldots \rangle$ is used to observe which cells are updated when. If the CA rule and an update pattern with an initial state is given, the state transitions for the ACA can be identified. A partial state transition diagram of four-cell rule 60 ACA with a null boundary condition is shown in Figure 1. The states are noted in circles, whereas the cells updated during state transitions are noted over the arrows. The update pattern for this transition $U = \langle \{2\}, \{1, 4\}, \{2, 4\}, \{3\}, \{2, 4\}, \{3, 4\}, \{2\}, \ldots \rangle$ is associated with CA state 15. The output of the first cell is considered as the least significant bit (LSB) of the CA state. It is, therefore, obvious that the state transition of ACAs depends on both the CA rule and the update pattern. However, a single state transition diagram may not cover all the CA states. To observe the transitions of other CA states, another one or more update


https://doi.org/10.25088/ComplexSystems.21.1.71

patterns may be needed. A set of update patterns can actually illustrate the transition of all states.

We address the reversibility issue for such ACAs in Section 3.

## 3. The Reversibility of Asynchronous Cellular Automata

The state transition diagram classifies the CA states as *cyclic* and *acyclic*. If a CA state lies on some cycle in the state transition diagram of the CA, the state is cyclic; otherwise, it is acyclic. The CAs are *reversible* if all the CA states are cyclic; otherwise, they are *irreversible*. The reversibility, explored in synchronous domain, guarantees that each CA state has a unique predecessor and successor.

**Definition 1**. The ACAs are reversible if each CA state can uniquely be reached starting from that particular state with an update pattern. Otherwise, they are irreversible.

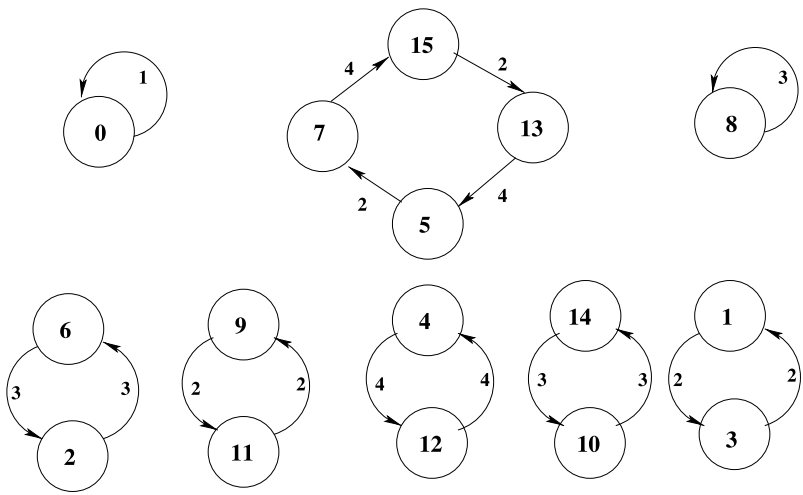


**Figure 2**. Four-cell rule 60 reversible ACAs in null boundary condition. The cells updated are noted on the edges.

Figure 2 depicts the state transition diagram of four-cell rule 60 reversible ACAs with null boundary condition. There are eight update patterns, one for each cycle, in the ACAs. The update patterns (with corresponding initial states) are ⟨{1}⟩ (0), ⟨{2}, {4}, {2}, {4}⟩ (15), ⟨{3}⟩ (8), ⟨{3}, {3}⟩ (6), ⟨{2}, {2}⟩ (9), ⟨{4}, {4}⟩ (4), ⟨{3}, {3}⟩ (14), and ⟨{2}, {2}⟩ (1). The CA rules, building blocks of reversible and irreversible ACAs, are classified as the reversible and irreversible rules.


https://doi.org/10.25088/ComplexSystems.21.1.71

**Definition 2**. A CA rule $R$ is an irreversible rule if there is a CA state that can never be cyclic for any update pattern, while the ACAs are configured with $R$. Otherwise, $R$ is a reversible rule.

For example, rule 77 (01001101) in null-boundary condition is an irreversible rule. Starting from the all-0 CA state, returning back to the 00 … 0 state in rule 77 ACAs cannot be done with any update pattern. On the other hand, rule 60 is a reversible rule in null and periodic boundary conditions. Each state can be uniquely reached for some update pattern (Figure 2).

Now we characterize the irreversible rules that can never configure reversible ACAs. Theorem 1 characterizes the irreversible rules in periodic boundary condition.

**Theorem 1**. A rule $R$ is irreversible if and only if the all-0 or all-1 state of the ACA, configured with $R$ in periodic boundary condition, is acyclic for all possible update patterns.

*Proof*. If the all-0 or all-1 state of the ACA, configured with $R$, cannot be returned back with any update pattern, then obviously the ACA and hence the $R$ are irreversible. Now, we shall show that $R$ is irreversible only if the all-0 or all-1 state is acyclic.

A CA state can be viewed as a sequence of RMTs. For example, the state 1100 in periodic boundary condition can be viewed as 3641, where 3, 6, 4, and 1 are corresponding RMTs on which the state can be changed. Combine the eight RMTs into four sets: {0, 2}, {1, 3}, {4, 6}, and {5, 7}. The three-bit binary representation of the RMTs shows that the middle bit of each set is the complement of each other. We next show that if a sequence of RMTs of an arbitrary rule, corresponding to some CA state, contains both the elements of any one of the above sets, the state is cyclic.

Consider that RMTs 0 and 2 are simultaneously present in a sequence of RMTs, corresponding to some CA state, $\mathcal{S}$. If RMT 0 is 0 or RMT 2 is 1, $\mathcal{S}$ can be updated properly to get a single-length cycle. If RMT 0 is 1 and RMT 2 is 0, then a two-length cycle can be designed by updating a single cell. Therefore, $\mathcal{S}$ is cyclic for any value of RMT 0 and RMT 2. If $\mathcal{S}$ contains the RMTs 1 and 3, 4 and 6, or 5 and 7 simultaneously, then it can also be shown with similar logic that $\mathcal{S}$ is cyclic.

The rest of the states whose corresponding RMTs are from different sets may form single-length cycles depending on the RMT values by updating a single cell. The states that are not in some cycle can form two-length cycles by updating two or more consecutive cells. Hence, these states are also cyclic.

Therefore, all the states other than all-0 and all-1 of any ACA can be cyclic for some update patterns. Hence, if all-0 and all-1 states are cyclic, the rule $R$ that configures ACAs is reversible; otherwise, $R$ is irreversible. □


https://doi.org/10.25088/ComplexSystems.21.1.71

**Corollary 1**. A rule $R$ is irreversible if (*i*) the RMTs 0, 2, 7, and either RMT 3 or 6 of $R$ are 1, or (*ii*) the RMTs 0, 5, 7, and either RMT 1 or 4 are 0 in periodic boundary condition.

*Proof*. We shall prove the corollary by identifying the RMTs of $R$ for which the all-0 or all-1 state cannot be returned back (Theorem 1).

If RMT 0 is 1, the ACA, configured with $R$ in periodic boundary condition, cannot form a single-length cycle with an all-0 state because the next state contains at least one 1 while the ACAs are updated. To form a cycle, these 1s are to be 0 in subsequent steps. However, these 1s cannot be 0 if RMTs 2 and 7, and any one of RMTs 3 and 6 of $R$, are 1. Therefore, the all-0 state cannot be returned back if the RMTs 0, 2, 7, and either RMTs 3 or 6 of $R$ are 1.

Similarly, the ACA cannot form a single-length cycle with an all-1 state if RMT 7 is 0. Moreover, the ACA with the state can never form a cycle of any length in periodic boundary condition if RMTs 0 and 5, and any one of RMTs 1 and 4 of $R$, are 0. Hence, an all-1 state cannot be returned back if the RMTs 0, 5, 7, and either RMT 1 or 4 are 0. □

There are (*i*) 24 rules where RMTs 0, 2, 7, and either RMT 3 or 6 are 1, and (*ii*) another 24 rules where RMTs 0, 5, 7, and either RMT 1 or 4 are 0. The list of 48 such irreversible rules are noted in Table 2. The rest are reversible rules, each of which can configure reversible ACAs in periodic boundary condition for some update patterns. Now, we present Theorem 2 to characterize the irreversible rules in null boundary condition.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
| 16 | 20 | 24 | 28 | 64 | 66 | 68 | 70 |
| 72 | 74 | 76 | 78 | 80 | 84 | 88 | 92 |
| 141 | 143 | 157 | 159 | 173 | 175 | 189 | 191 |
| 197 | 199 | 205 | 207 | 213 | 215 | 221 | 223 |
| 229 | 231 | 237 | 239 | 245 | 247 | 253 | 255 |

**Table 2**. Irreversible rules in periodic boundary condition.

**Theorem 2**. A rule $R$ is irreversible if and only if the all-0, all-1, or 10101…1 state of ACAs, configured with $R$ in null boundary condition, is acyclic for all possible update patterns.

*Proof*. The proof is similar to that of Theorem 1 with an exception that the 10101…1 state is to be considered in case of null boundary condition. Irrespective of $R$, this state cannot be cyclic in null boundary condition. □


https://doi.org/10.25088/ComplexSystems.21.1.71

**Corollary 2.** A rule $R$ is irreversible if (*i*) the RMTs 0 and 2, and either RMT 3 or 6 of $R$, are 1, or (*ii*) the RMTs 0, 1, 3, 4, 5, 6, and 7 are 0, or (*iii*) the RMTs 0 and 2 are 0, RMTs 5 and 7 are 1, and if RMT 1 is 0 or RMT 3 is 1, then either RMT 4 is 0 or RMT 6 is 1, while $R$ configures an ACA in null boundary condition.

*Proof.* We shall prove the corollary by identifying the RMTs of $R$ for which the all-0, all-1, or 10101…1 state cannot be returned back (Theorem 2).

If RMT 0 is 1, the ACA, configured with $R$ in null boundary condition, cannot form a single-length cycle with the all-0 state, as the next state always contains at least one 1. To form a cycle, these 1s are to be 0 in subsequent steps. However, these 1s cannot be 0 if RMTs 2 and any one of RMTs 3 and 6 of $R$ are 1. Therefore, the all-0 state cannot be returned back if the RMTs 0, 2, and either RMT 3 or 6 of $R$ are 1.

In null boundary condition, the state of the left (right) neighbor of the leftmost (rightmost) cell is always 0. So, RMT 3 and RMT 7 (RMT 6 and RMT 7) of $R$ are equivalent for the leftmost (rightmost) cell. Therefore, the ACA with all-1 state can form a single-length cycle for some update pattern if RMT 3, RMT 6, or RMT 7 is 1. To restrict such a cycle, the RMTs 3, 6, and 7 of $R$ are 0. While these RMTs are 0, the ACA with all-1 state, due to the update of cells, reaches to another state that contains at least one 0. However, the all-1 state cannot be returned back if RMTs 0, 1, 4, and 5 are 0. So, the all-1 state is acyclic if the RMTs 0, 1, 3, 4, 5, 6, and 7 are 0.

To form a single-length cycle with the 10101…1 state, RMT 2 is to be 1 or RMT 5 is to be 0. If RMT 2 is 0 and RMT 5 is 1, a single-length cycle in null boundary condition cannot be formed with the 10101… 1 state. However, two or more consecutive bits of the derived state may be 0 or 1; even all-0 or all-1 states may be reached. The 10101… 1 state cannot be returned back from the all-1 state in null boundary condition if RMT 3 and 6 are 1, and from the all-0 state if RMT 0 is 0. For any combination of 0s and 1s in the derived state, if any of the following RMT values are found in $R$, the 10101… 1 state cannot be returned back.

| 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|---|---|---|---|---|---|---|---|
| (7) | (6) | (5) | (4) | (3) | (2) | (1) | (0) |
| 1 | $*$ | 1 | 0 | 1 | 0 | $*$ | 0 |
| 1 | 1 | 1 | $*$ | 1 | 0 | $*$ | 0 |
| 1 | $*$ | 1 | 0 | $*$ | 0 | 0 | 0 |
| 1 | 1 | 1 | $*$ | $*$ | 0 | 0 | 0 |

The RMTs that can take any value (0/1) are denoted with "$*$". Hence, the 10101… 1 state is acyclic if the RMTs 0 and 2 are 0, RMTs 5 and


https://doi.org/10.25088/ComplexSystems.21.1.71

7 are 1, and if RMT 1 is 0 or RMT 3 is 1, then either RMT 4 is 0 or RMT 6 is 1. □

In null boundary condition, there are (*i*) 48 irreversible rules while RMTs 0, 2, and either 3 or 6 of *R* are 1, (*ii*) two irreversible rules while RMTs 0, 1, 3, 4, 5, 6, and 7 are 0, and (*iii*) nine irreversible rules while RMTs 0 and 2 are 0, RMTs 5 and 7 are 1, and if RMT 1 is 0 or RMT 3 is 1, then either RMT 4 is 0 or RMT 6 is 1. Such irreversible rules are listed in Table 3. The rest are reversible rules, which can configure reversible ACAs in null boundary condition with some update patterns.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 4 | 13 | 15 | 29 | 31 | 45 | 47 |
| 61 | 63 | 69 | 71 | 77 | 79 | 85 | 87 |
| 93 | 95 | 101 | 103 | 109 | 111 | 117 | 119 |
| 125 | 127 | 141 | 143 | 157 | 159 | 160 | 168 |
| 170 | 173 | 175 | 189 | 191 | 197 | 199 | 205 |
| 207 | 213 | 215 | 221 | 223 | 224 | 229 | 231 |
| 232 | 234 | 237 | 239 | 240 | 245 | 247 | 248 |
| 250 | 253 | 255 | | | | | |

**Table 3.** Irreversible rules in null boundary condition.

However, the reversibility of ACAs depends not only on the rule, but also on update patterns. For example, rule 60 can configure irreversible ACAs (Figure 1) as well as reversible ACAs (Figure 2) depending upon the update patterns. Since the ACA cells are independent, and so updated arbitrarily, it cannot be predicted in advance that ACAs configured with a reversible rule are reversible. If a set of update patterns received from ACAs configured with a reversible rule during generation of all states are given, then only whether the ACAs were reversible can be analyzed. This discussion leads to Theorem 3.

**Theorem 3.** It is hard to synthesize reversible one-dimensional ACAs.

However, the update patterns can be designed for the cycles of some reversible ACAs. While the ACAs follow those update patterns, cycles are formed. We identify such update patterns in Section 4.

## 4. Identifying the Update Pattern for a Cycle

The reversible rules require different sets of update patterns to get reversible ACAs. Even for a particular reversible rule, various sets of update patterns may be identified that result in different reversible ACAs. An update pattern can produce a cycle if the initial state and the ACA are given. In this section, we identify such an update pattern that forms a cycle for some reversible ACAs. We next present a theorem that characterizes the states forming a cycle.


https://doi.org/10.25088/ComplexSystems.21.1.71

**Theorem 4**. The sequence of unique states $\langle S_1, S_2, \ldots, S_l, S_1\rangle$ of an $n$-cell CA forms a cycle of length $l \geq 1$ if the number of bits that flip at the $i^{\text{th}}$ $(1 \leq i \leq n)$ position of the states is either 0 or even.

*Proof.* Consider that the $i^{\text{th}}$ bit of the CA state $S_1$ is $d$. Now, if the $i^{\text{th}}$ bit position of the sequence is flipped to $d'$ in some $S_j$, then the bit position is to be flipped in some $S_{j+k}$ to get back $d$ at the $i^{\text{th}}$ bit position, where $1 < j < j + k \leq l + 1$. So, two transitions are there. If another such $j$ exists, then corresponding $k$ also exists. Hence, an even number of bit flipping is required. □

To get a cycle for some reversible ACAs, an update pattern along with some initial state is required that generates $l$ distinct CA states for a cycle of length $l$. Since the states of a cycle are to be distinct, the update pattern should be designed in such a way that at least one bit of a state flips to get the next state. Moreover, in any subsequence of states, the bits of states are not to be flipped an even number of times (Theorem 4). If they flip, the $l$ states cannot be distinct.

Therefore, generation of distinct states depends not only on the update pattern, but also on the initial state. This is because the initial state may not allow an arbitrary bit to flip for an arbitrary reversible rule that configures the ACA. However, rule 51 (Table 1) is the only rule that always allows a cell to flip its state when updated. So, rule 51 ACAs do not depend on the initial state to form a cycle. The following rule is designed to generate an update pattern for a cycle of length $2^i$ $(1 \leq i \leq n)$ by updating a single cell at a time, where $n$ is the number of ACA cells.

> To get a cycle of length $2^i$ $(1 \leq i \leq n)$ of an $n$-cell rule 51 ACA, form a sequence of $i$ cells to be updated arbitrarily. Start with an arbitrary state. Update the $\left(2^{j-1}\right)^{\text{th}}$ state by updating the $j^{\text{th}}$ cell $(1 \leq j \leq i)$ of the sequence to generate the next state. Repeat the update of the $j^{\text{th}}$ cell after each $2^j$ state, where $j < i$. However, update the $i^{\text{th}}$ cell again after the $2^{i-1}$ state to get a cycle of length $2^i$.

**Example 1**. To design a full-length cycle for a four-cell rule 51 ACA (length $= 2^4$), all the cells are to be updated in some sequence. Consider that the sequence of updating is SEQ $= \langle 1, 2, 3, 4\rangle$ and the initial state is 0100. Each $j^{\text{th}}$ cell of SEQ is selected for the first time to update the $\left(2^{j-1}\right)^{\text{th}}$ state. Hence, to get the second state, the first bit of the initial state ($\left(2^{j-1}\right)^{\text{th}}$ state, where $j = 1$) is updated. Similarly, the second, third, and fourth cells are selected for the first time to update the second, fourth, and eighth states, respectively. The first cell is again selected to update the third, fifth, and all odd states (i.e., after each $2^j$ state where $j = 1$). After the first time update, the second and


https://doi.org/10.25088/ComplexSystems.21.1.71

third cells are selected repeatedly to update after every $2^2$ and $2^3$ states, respectively. The last cell is updated for the second time after $2^3$ states ($2^{i-1}$ states where $i = 4$) to complete the cycle. Therefore, the sequence of states in the cycle is ⟨0100, 1100, 1000, 0000, 0010, 1010, 1110, 0110, 0111, 1111, 1011, 0011, 0001, 1001, 1101, 0101, 0100⟩. The update pattern is ⟨{1}, {2}, {1}, {3}, {1}, {2}, {1}, {4}, {1}, {2}, {1}, {3}, {1}, {2}, {1}, {4}⟩ (Figure 3(a)). Here, the update pattern is independent of the initial state, but depends on SEQ (the update pattern and the cycle of rule 51 ACA are the same for both the boundary conditions). However, if the cells are updated randomly, the ACA may not even be reversible. No cycle can be found in such a case (Figure 3(b)).

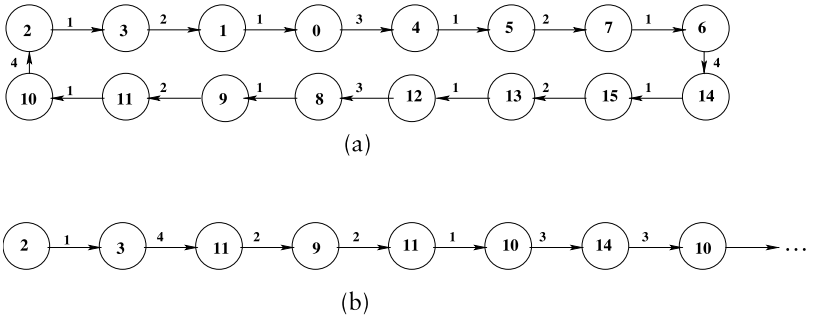


**Figure 3**. State transition of four-cell rule 51 ACAs (updating a single cell in each step). Here, the output of the first cell is considered as the LSB. (a) Full-length cycle ACAs. (b) Random update of cells.

However, cycles can be formed by updating multiple cells simultaneously. An $n$-cell rule 51 ACA can form a cycle of maximum length $2^{n-m+1}$ while $m$ cells ($1 \le m \le n$) are updated simultaneously. In such a case, the same way of single-cell update to get a cycle can be followed with an exception that each entry in the sequence of cells, to be updated, is a set of $m$ cells. Example 2 illustrates the cycle formation by updating multiple cells.

**Example 2**. Let us consider that $n = 4$ and $m = 2$. To get an eight-length ($2^{n-m+1}$) cycle of the ACA, a sequence SEQ = ⟨{1, 2}, {2, 3}, {3, 4}⟩ of cells is formed arbitrarily. Consider that the initial state is 0100. The first and second bits are updated to generate the second state (1000). Similarly, the cells of the second and third entries of SEQ are selected to update the second and fourth states. As in Example 1, the cells of the first set ({1, 2}) are repeatedly selected to update the odd states. The cells of the second set ({2, 3}) are selected again to update the sixth state. Therefore, a sequence ⟨0100, 1000, 1110,


https://doi.org/10.25088/ComplexSystems.21.1.71

0010, 0001, 1101, 1011, 0111, 0100⟩ of states is obtained and the update pattern is ⟨{1, 2}, {2, 3}, {1, 2}, {3, 4}, {1, 2}, {2, 3}, {1, 2}, {3, 4}⟩ (Figure 4(a)). However, while two cells are arbitrarily updated (violating the given rule), no such cycle is formed (Figure 4(b)).

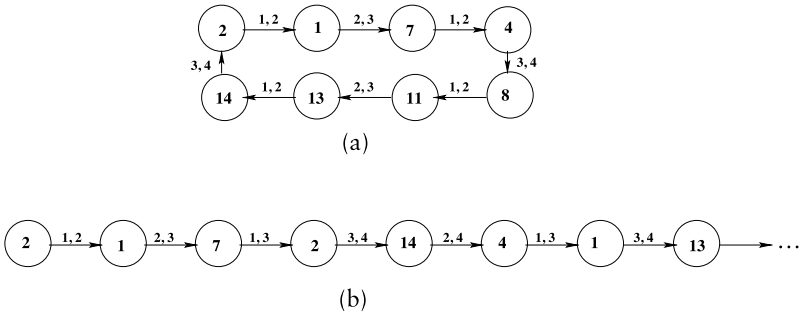


**Figure 4**. State transition of four-cell rule 51 ACAs, updating two cells in each step (the output of the first cell is considered as the LSB). (a) Eight-length cycle of rule 51 ACAs. (b) Random update of cells.

The update method, designed for the rule 51 reversible ACA, guides us to develop Algorithm 1, which finds the update pattern for a cycle of some reversible ACAs. The algorithm is independent from the boundary condition. It takes the CA rule, the cycle length to be designed ($2^i$), the initial state ($S$), and the number of cells updated in a single step ($m$) as input. However, with arbitrary ACAs and an arbitrary initial state, a cycle of given length may not be designed. In such cases, the algorithm finds a cycle that is close in length with the given cycle length. It outputs the update pattern with the cycle length, if the cycle can be designed.

The algorithm first forms a sequence of $i$ unique sets arbitrarily. The sets are also designed arbitrarily with $m$ ACA cells per set. The update style of rule 51 reversible ACAs is followed to generate the update pattern. If no bit flips during the update of a set of $m$ cells, another set of $m$ cells is searched so that at least one bit flips. If no such set is found, then the algorithm reports that "Cycle is not possible". While $2^i$ states are covered but no cycle is formed, the algorithm attempts to form a cycle by generating a very few states.

**Algorithm 1**. FindACACycle

**Input:** $R$ (rule), $n$ (# cells), $2^i$ (cycle length, $1 \le i \le n$), $S$ (initial state), $m$ (# cells updated in each step)

**Output:** Update pattern with the cycle length, if a cycle is possible

*Step 1:* Form a sequence SEQ of $i$ unique sets of $m$ ACA cells arbitrarily.


https://doi.org/10.25088/ComplexSystems.21.1.71

*Step 2:* Load the ACA, configured with $R$, with $S$.

*Step 3:* For $k = 1$ to $2^i$, repeat Step 4 to Step 9.

*Step 4:* If $k = 2^{j-1}$ $(1 \le j \le i)$, select the $j^{\text{th}}$ set of SEQ.
If $k = 2^i$, select the $i^{\text{th}}$ set of SEQ.
If $k = 2^{j-1} + p * 2^j$ ($p$ is a positive integer and $1 \le j < i$), select the $j^{\text{th}}$ set.

*Step 5:* Update ACA cells of the selected set.

*Step 6:* If no cell flips during the update, find a set of $m$ cells so that

(a) at least one cell flips, and (b) the generated state is unique.

Otherwise, go to Step 9.

*Step 7:* If no such set is found in Step 6, go to Step 14.

*Step 8:* Update the ACA cells according to the set designed in Step 6.

*Step 9:* Print the ACA cells that are updated to generate the next state of $k$.

*Step 10:* If no cycle is formed, identify the bits of the $2^i + 1$ state that differ from the initial state, $S$. Otherwise, go to Step 15.

*Step 11:* Update the ACA cells to flip the identified bits.

*Step 12:* If few cells flip, print those cells. Update the nearest cells of the remaining bits (one-by-one or more than one at a time) so that the $S$ is reached within a few steps.

*Step 13:* If a cycle is formed, go to Step 15.

*Step 14:* Print "Cycle is not possible" and exit.

*Step 15:* Print the length of the cycle and exit.

Example 3 illustrates the execution of Algorithm 1.

**Example 3**. Let us consider $R = 123$, $n = 6$, cycle length $= 8$ $(2^3)$, $S = 011111$, and $m = 2$. The formation of the cycle following Algorithm 1 is shown in Figure 5. First, a sequence of three sets SEQ $= \langle\{1, 3\}, \{1, 4\}, \{2, 4\}\rangle$ is formed arbitrarily (Step 1). The ACA is configured with rule 123 in null boundary condition. To get the next state of 011111 (initial state), the first and third cells are updated (Steps 4 and 5). In Figure 5, the update pattern of a rule 51 ACA is noted on the left side of the states, and the update pattern generated by the algorithm is shown on the right side. To update the second state (similar to the sixth state) according to the update pattern of the rule 51 ACA, the set {1, 4} is selected. Since no cell flips here, another set {1, 5} is searched (Step 6). After the generation of eight states, a cycle is not formed. Another four states are generated to form a cycle (Steps 10–12). Therefore, the length of the cycle is 12.


https://doi.org/10.25088/ComplexSystems.21.1.71

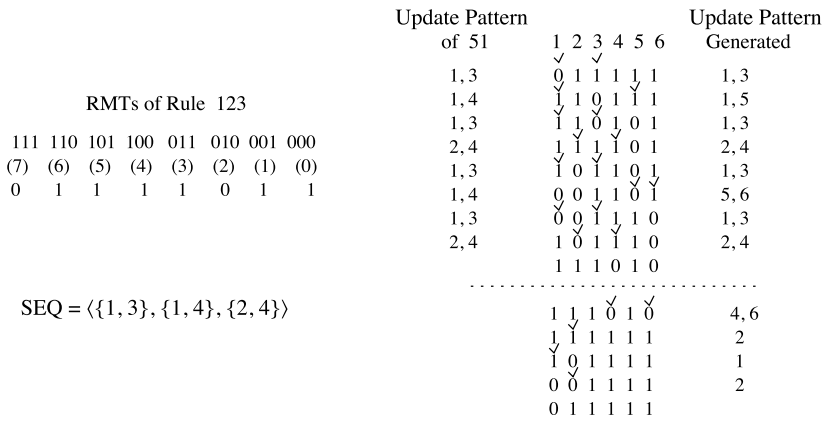


**Figure 5**. Generation of the cycle for the rule 123 ACA. At most, two cells are updated simultaneously.

We have experimented with different reversible rules. It is found that for a number of reversible rules, the update pattern can be designed utilizing Algorithm 1 to get a full-length cycle (by updating a single cell at a time). A few of such rules are: 3, 19, 35, 83, 115, 131, 147, 163, 179, 211, and 243.

## 5. Conclusion

The reversibility in one-dimensional asynchronous cellular automata (ACAs) has been addressed in this paper. The ACA cells are updated independently. Depending on their update during state transition, the update pattern is defined. The paper has classified the cellular automata (CAs) rules as reversible and irreversible. The irreversible rules cannot configure reversible ACAs with any set of update patterns. The reversibility of ACAs depends on both the rule and update patterns. Finally, the paper reports an algorithm to get an update pattern for a cycle of ACAs.

## Acknowledgment

This work is supported by AICTE Career Award Fund (F.No. 1-51/RID/CA/29/2009-10).


https://doi.org/10.25088/ComplexSystems.21.1.71

## References

[1] J. von Neumann, *The Theory of Self-Reproducing Automata* (A. W. Burks, ed.), Urbana, IL: University of Illinois Press, 1966.

[2] S. Wolfram, *Theory and Applications of Cellular Automata*, Singapore: World Scientific, 1986.

[3] T. E. Ingerson and R. L. Buvel, "Structure in Asynchronous Cellular Automata," *Physica D: Nonlinear Phenomena*, **10**(1–2), 1984 pp. 59–68. doi:10.1016/0167-2789(84)90249-5.

[4] R. Cori, Y. Metivier, and W. Zielonka, "Asynchronous Mappings and Asynchronous Cellular Automata," *Information and Computation*, **106**(2), 1993 pp. 159–202. doi:10.1006/inco.1993.1052.

[5] N. Fatès, E. Thierry, M. Morvan, and N. Schabanel, "Fully Asynchronous Behavior of Double-Quiescent Elementary Cellular Automata," *Theoretical Computer Science*, **362**(1–3), 2006 pp. 1–16. doi:10.1016/j.tcs.2006.05.036.

[6] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[7] S. Amoroso and Y. N. Patt, "Decision Procedures for Surjectivity and Injectivity of Parallel Maps for Tessellation Structures," *Journal of Computer and System Sciences*, **6**(5), 1972 pp. 448–464. doi:10.1016/S0022-0000(72)80013-8.

[8] S. Das and B. K Sikdar, "Classification of CA Rules Targeting Synthesis of Reversible Cellular Automata," in *Proceedings of the 7th International Conference on Cellular Automata for Research and Industry (ACRI06)*, Perpignan, France (S. El Yacoubi, B. Chopard, and S. Bandini, eds.), Berlin: Springer-Verlag, 2006, pp. 68–77. doi:10.1007/11861201_11.

[9] T. Toffoli, "Computation and Construction Universality of Reversible Cellular Automata," *Journal of Computer and System Sciences*, **15**(2), 1977 pp. 213–231. doi:10.1016/S0022-0000(77)80007-X.

[10] J. Lee, F. Peper, S. Adachi, K. Morita, and S. Mashiko, "Reversible Computation in Asynchronous Cellular Automata," in *Proceedings of the 3rd International Conference on Unconventional Models of Computation (UMC02)*, Kobe, Japan (C. Calude, M. J. Dinneen, F. Peper, eds.), London: Springer-Verlag, 2002, pp. 220–229.

[11] C. L. Nehaniv, "Asynchronous Automata Networks Can Emulate Any Synchronous Automata Network," *International Journal of Algebra and Computation*, **14**(5–6), 2004 pp. 719–739. doi:10.1142/S0218196704002043.


https://doi.org/10.25088/ComplexSystems.21.1.71