Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computer and System Sciences 70 (2005) 201–220

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

Delay-insensitive computation in asynchronous cellular automata

Jia Lee*, Susumu Adachi, Ferdinand Peper, Shinro Mashiko

National Institute of Information and Communications Technology, Nanotechnology Group, 588-2 Iwaoka, Iwaoka-cho, Nishi-ku, Kobe 651-2492, Japan

Received 6 June 2004; received in revised form 13 October 2004

Available online 9 December 2004

Abstract

Asynchronous cellular automata (ACA) are cellular automata that allow cells to update their states independently at random times. Because of the unpredictability of the order of update, computing on ACA is usually done by simulating a timing mechanism to force all cells into synchronicity after which well-established synchronous methods of computation can be used. In this paper, we present a more effective method of computation based upon a 4-state two-dimensional ACA with von Neumann neighborhood that is based on the construction in the cellular space of delay-insensitive circuits, a special type of asynchronous circuits, whose operations are robust to arbitrary delays in operators or interconnection lines. We show that this novel ACA model can be used to construct a universal Turing machine, which suffices to prove its computational universality.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Cellular automata; Asynchronous; Delay-insensitive circuits

1. Introduction

A *cellular automaton* (CA) is a discrete dynamical system consisting of an *d*-dimensional array of identical finite-state automata (cells) ($d \geqslant 1$). Each cell is connected uniformly to a neighborhood of a finite number of cells, and has a state from a finite state set. It updates its state according to a transition function, which determines a cell's state based on the states of the cells in its neighborhood. In most CA models, there is a special state called *quiescent* state, in which a cell will never change its state as long as all the cells in its neighborhood are quiescent.

* Corresponding author.
E-mail address: lijia@crl.go.jp, lijia@nict.go.jp (J. Lee).

Von Neumann [26] was the first to conceive a cellular automaton capable of universal computation as well as universal construction. His CA model is composed of a two-dimensional array of an infinite number of cells, in which each cell can update its state at the next time according to its own state and the current states of the four non-diagonal cells adjacent to it (so-called Von Neumann neighborhood). von Neumann used 29-states per cell to construct a universal Turing machine, in which all cells undergo state transitions at the same time [26]. After this, Codd [6], Banks [3] and Serizawa [23] reduced the number of cell states to 8, 4 and 3, respectively, with the universality maintained, using the same framework as von Neumann's universal CA. Banks also designed a 2-state CA with von Neumann neighborhood to realize any synchronous circuits, implying computational universality [3].

All the above universal CA require the state transitions of each cell to be synchronized by a central clock, so they are called synchronous cellular automata. As a generalized model of CA [17], asynchronous cellular automata (ACA) allow cells to evolve their state transitions independently at random times, thereby erasing the need to provide a central clock for the entire cellular space. However, the absence of a central clock in ACA may cause unexpected behaviors of cells during computation (see e.g. [10,27]). A conventional method of computing on ACA simulates a timing mechanism that forces all cells into synchronicity [5,7,9,13,16–18,24,25], which allows the use of well-developed synchronous methods for computation.

Although any n-state d-dimensional synchronous CA with a symmetric neighborhood (e.g., von Neumann neighborhood) can be simulated by an (n^2+2n)-state d-dimensional ACA with the same neighborhood [13], computing can be done directly on ACA in an asynchronous way, rather than using synchronous computational methods, by embedding so-called *delay-insensitive* (DI) circuits in asynchronous cellular automata [2,12,20,21] (see also [8]). A DI-circuit is a special type of asynchronous circuit whose correctness of operation is not affected by arbitrary delays in operators or interconnection lines [11,14,19]. In particular, a 5-state ACA with von Neumann neighborhood was presented in [12], by which any arbitrary DI-circuit can be constructed. Although this result shows the computational universality of the 5-state ACA [12], it remains unclear what the minimal number of cell states should be for an ACA with von Neumann neighborhood to be capable of universal computation.

In this paper, we introduce a novel 4-state asynchronous cellular automaton with von Neumann neighborhood, which uses rotation- and reflection-symmetric transition rules to describe the interactions between cells. Instead of implementing the primitive operators [14] on the ACA to realize any arbitrary DI-circuit like in [2,12,20], we design a particular circuit [15] in our asynchronous cellular automaton to simulate a universal Turing machine, which implies the computational universality of our 4-state ACA.

This paper is organized as follows. Section 2 provides some basic notions and known results on DI-circuits. Our novel 4-state ACA with von Neumann neighborhood is demonstrated in Section 3, followed by the conclusion in Section 4.

2. Delay-insensitive circuits

A delay-insensitive (DI) circuit is an asynchronous circuit in which signals may be subject to arbitrary delays but without these being an obstacle to its correct operation. The circuit is composed of interconnection lines and modules. Signals are transmitted along the lines and are processed by the modules.

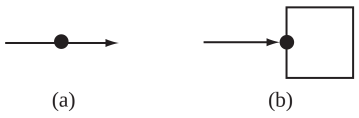


Fig. 1. (a) Transfer of a signal on a line; (b) a pending signal, in which the block denotes a module.

Definition 1. A *sequential machine* is defined as $N = (Q, q_0, \Sigma, \Delta, f, g)$, where Q is a finite set of states, q_0 is the initial state, Σ is the set of input symbols, Δ is the set of output symbols, $f : Q \times \Sigma \to Q$ is the state-transition function, and $g : Q \times \Sigma \to \Delta$ is the output function.

Definition 2. A *module* is defined as (I, O, N), where I is a finite set of input lines, O is a finite set of output lines, and $N = (Q, q_0, \Sigma, \Delta, f, g)$ is a sequential machine with Σ in one-to-one correspondence with I, and Δ in one-to-one correspondence with 2^O.

Definition 3. A *signal* is the change of state or value of a line, and is transferred on a line from an output of a module to an input of another module. Being one-valued, signals are denoted by a token on a line as in Fig. 1(a). A *pending signal* is a signal that has arrived on an input line of a module, but will not give rise to output of the module unless other signals arrive on different input lines. A pending signal is denoted as in Fig. 1(b).

Definition 4. A circuit is *delay-insensitive* if its external input and output behavior remains unchanged, regardless of arbitrary delays in any modules or interconnection lines.

Keller [11] formulated several operating conditions to characterize the class of DI-circuits, under which any circuit can be realized by a fixed set of primitive modules. Keller's conditions are summarized as follows.

Condition 1: The input and output lines of a module are fixed and separated.

Condition 2: A line is only allowed to connect at most two modules to its ends. Also, a line is the input to exactly one module, and the output to exactly one module.

Condition 3: Once a module outputs a signal, it cannot withdraw the signal from the output line.

Condition 4: If two signals arrive at different input lines simultaneously, then the action may be chosen arbitrarily by the module, as if one signal, then the other, appeared in an order specified by the module. This is called the *arbitration* condition.

Condition 5: A module may be subject to arbitrary (but finite) delays between the assimilation of input signals and the production of the corresponding output signals.

Condition 6: A signal on a line must eventually be assimilated by the module at its destination. The signal may thereby undergo arbitrary (but finite) delays before its arrival.

Condition 7: If there is a signal on an input line of a module, then it must be assimilated before the next signal can be put on the same line.

Definition 5. A module is *serial* if every signal on one of its input lines, must be followed by exactly one signal on an output line of the module, before the next signal can be assimilated by the same module. Otherwise, it is a *parallel* module.

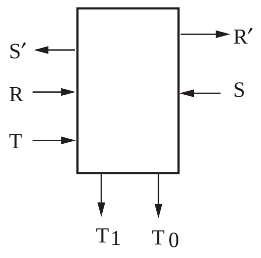


Fig. 2. An S module.

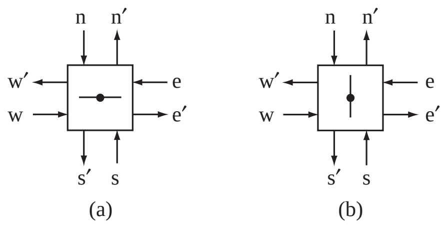


Fig. 3. (a) An RE in state 0, and (b) an RE in state 1.

An example of a serial module is the so-called **S** (Select module)[11]: This module has two states, 0 and 1, the latter being the initial state (see Fig. 2). A signal on input line $S(R)$ always sets the state to 1(0) and is transferred to $S'(R')$. A signal on input line T (test signal) does not change the internal state, but is transferred to T_1 when the module is in state 1, and to T_0 if it is in state 0.

Another important serial module in this paper is Morita's [15] *Rotary Element* (RE): There are two states, 0 and 1, in an RE module (see Fig. 3). A signal arriving on input line $s(n)$ always sets the state to 1, and a signal on input line $e(w)$ always resets the state to 0. When the module is in state 1, a signal arriving on $s(n)$ is transferred to output line $n'(s')$, and an arrival on $e(w)$ is transferred to $n'(s')$. When the module in state 0, a signal arriving on $s(n)$ is transferred to output line $e'(w')$, and a signal on $e(w)$ is transferred to $w'(e')$. Simultaneous signals on any pair of input lines are not allowed.

The RE is capable of universal computation, in the sense that any (reversible) Turing machine is realizable by a circuit of RE modules, in which there is at most one signal moving around at any time [15]. Obviously, delays in any of the REs or lines do not affect the correct operation of the entire circuit, and hence, it is delay-insensitive. Thus, such circuits consisting of REs can operate asynchronously without the need of a central clock signal [15].

Definition 6. A set of primitive modules is *serial universal*, if each serial module can be constructed by the modules in the set.

Here, we employ three types of primitive modules defined as follows, which are serial universal according to [19] (see also [11]).

(1) MERGE: A signal on input line $I_1(I_2)$ in Fig. 4(a) is assimilated and output to O. Simultaneous signals on I_1 and I_2 are not allowed.
(2) FORK: A signal on input line I in Fig. 4(b) is assimilated and duplicated on output lines O_1 and O_2.

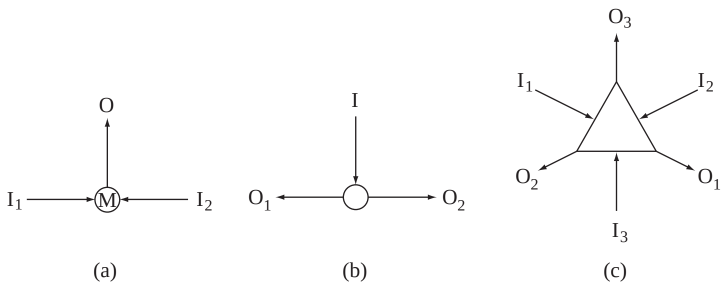


Fig. 4. A serial universal set of DI modules.

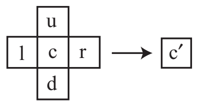
(3) TRIA: Signals on both I_x ($x \in \{1, 2, 3\}$) and I_y ($y \in \{1, 2, 3\}/\{x\}$) in Fig. 4(c) are assimilated and result in a signal output to O_{6-x-y}. Simultaneous signals on I_1, I_2 and I_3 are not allowed. A signal on only one of the input lines keeps pending until a signal on one of the other lines is received.

Theorem 1 (*Patra and Fussell [19]*). *{MERGE, FORK, TRIA} is serial universal.*

3. Computing asynchronously on a 4-state asynchronous cellular automaton with von Neumann neighborhood

In this section, we assume a CA model consisting of a two-dimensional array of cells, in which each cell has a neighborhood composed of its four non-diagonal adjacent cells along with itself (so-called von Neumann neighborhood) (Fig. 5). We present a novel 4-state ACA with von Neumann neighborhood for universal computation. We achieve this result by embedding delay-insensitive circuits in the 4-state asynchronous cellular space, which enables the construction of a Turing machine that can conduct universal computing tasks.

Definition 7. A *two-dimensional deterministic cellular automaton with von Neumann neighborhood* is defined as $A = (Z^2, V, f, v_0)$, where Z is a set of all integers, V is a finite set of states per cell ($V \neq \emptyset$). The mapping $f : V^5 \to V$ is called a *local function*, that determines the state transition of a cell, depending on the present states of the five neighborhood cells (i.e. center, upward, rightward, downward and leftward cells). Thus a *transition rule* $f(c, u, r, d, l) = c'$ can be depicted as follows.



The state $v_0(\in V)$ is a *quiescent* state satisfying $f(v_0, v_0, v_0, v_0, v_0) = v_0$. A *configuration* over V is a mapping $c : Z^2 \to V$, which assigns to each cell in A a certain state from V.

Definition 8. Let $A = (Z^2, V, f, v_0)$ be a CA with von Neumann neighborhood.

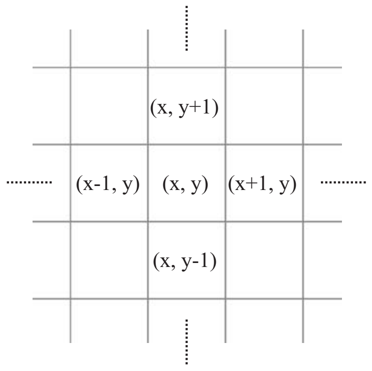


Fig. 5. Two-dimensional cellular space with von Neumann neighborhood.

(i) Transition rules in *A* are *rotation-symmetric*, iff

$$\forall (c, u, r, d, l) \in V,\ f(c, u, r, d, l) = f(c, l, u, r, d).$$

(ii) Transition rules in *A* are *reflection-symmetric*, iff

$$\forall (c, u, r, d, l) \in V,\ f(c, u, r, d, l) = f(c, u, l, d, r) = f(c, d, r, u, l).$$

The rotation- and reflection-symmetric transition rules specify an isotropic cellular space. According to the following definition [17] on the transitions of configurations in ACA, each cell in an ACA can update its state at random times independent of other cells.

Definition 9. Let $A = (Z^2,\ V,\ f,\ v_0)$ be a CA with von Neumann neighborhood, and $Conf(V)$ denote the set of all configurations over *V*, i.e. $Conf(V) = \{c \mid c : Z^2 \to V\}$.

(i) If *A* is a synchronous CA, then the function $F : Conf(V) \to Conf(V)$ defined as follows is called a *global function* of *A*.

$$\forall (x, y) \in Z^2,\ F(c)(x, y) = f(c(x, y), c(x, y+1), c(x+1, y), c(x, y-1), c(x-1, y))$$

(ii) If *A* is an asynchronous CA, then for any $c_i, c_j \in Conf(V)$ such that

$$\forall\, (x, y) \in Z^2,\ c_i(x, y) = c_j(x, y) \vee$$
$$(c_j(x, y) = f(c_i(x, y), c_i(x, y+1), c_i(x+1, y), c_i(x, y-1), c_i(x-1, y)),$$

there exists a transition from c_i to c_j written by $c_i \to c_j$. For any $c_s, c_t \in Conf(V)$, we denote $c_s \overset{+}{\to} c_t$ iff $c_s \to c_t$ or $\exists c'_s \in Conf(V),\ c_s \to c'_s \overset{+}{\to} c_t$.

Our 4-state ACA with von Neumann neighborhood is defined by $A_4 = (Z^2, \{0, 1, 2, 3\}, f_4, 0)$, which uses rotation- and reflection-symmetric transition rules to describe the interactions between cells. We use the notations □, ▨, ■ or ⊠ in the figures to represent a cell in the state 0, 1, 2 or 3, respectively. Transition rules in A_4 are given in Fig. 6, with their rotation- and reflection-symmetric equivalents left out. Combinations of states for which no transition rules are expressed in Fig. 6 are supposed to give rise to trivial transitions, i.e., transitions that do not change the states of cells.

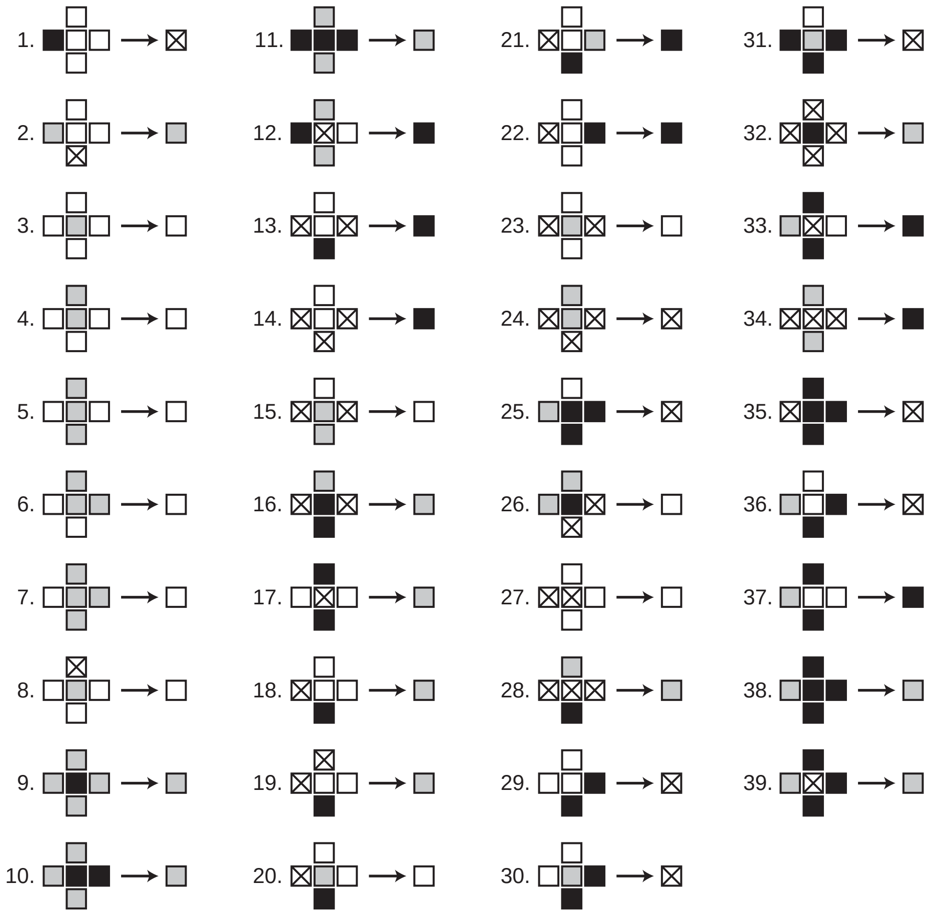


Fig. 6. List of transition rules in A_4.

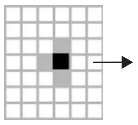


Fig. 7. A signal. The arrow indicates the direction in which the signal propagates.

3.1. *Configurations representing signals*

Signals are realized in the same way as in [12]. Fig. 7 gives the basic configuration of a signal.

A line between two modules in a DI-circuit is implemented as a path of quiescent cells. The signal in Fig. 7 contains one cell in state 2 which is the core of the signal, and three cells in state 1. The signal propagates into the direction in which a quiescent cell is adjacent to the core cell. The transition rules 1–12 in Fig. 6 are used for signal transmission.

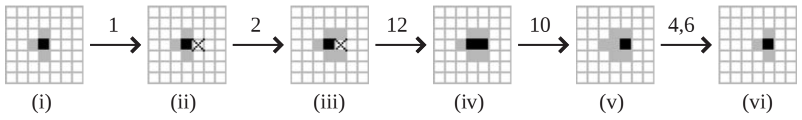


Fig. 8. Transmitting a signal on a path. The integers above each arrow denote the transition rules (or their rotation- and reflection-symmetric equivalents) in Fig. 6 used to change the states of the cells in the configurations.

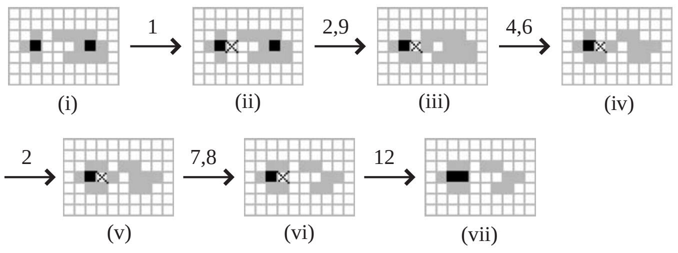


Fig. 9. Garbage left on the path blocks the transmission of a signal until it is cleaned up, but cannot interfere with the signal.

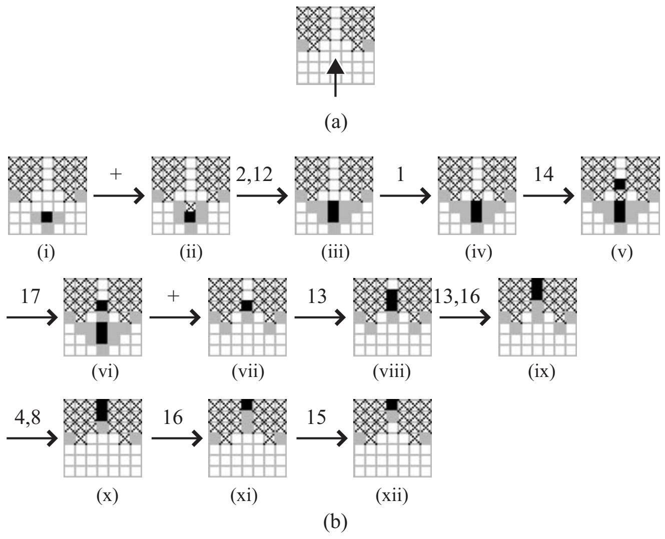
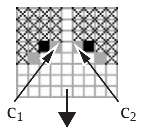
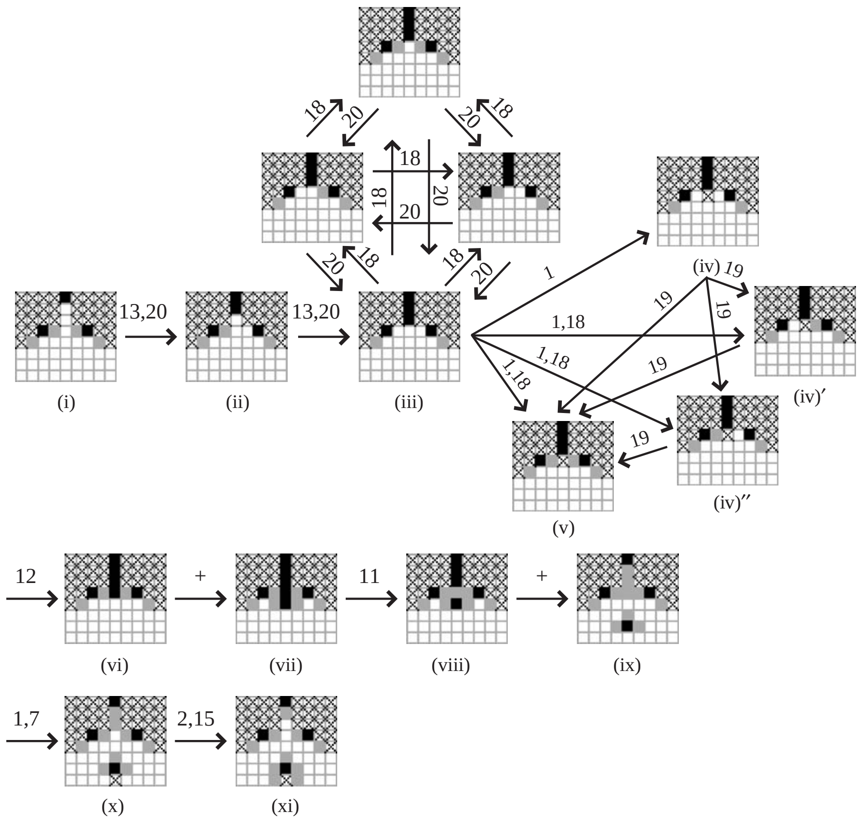


Fig. 10. (a) An Entrance. For convenience, we assume that the cellular space out of this local configuration is filled with quiescent cells; (b) entrance operating on an input signal.

The process of transmitting a signal on a path in A_4 can be summarized as follows (see also [12]): (1) A quiescent cell adjacent to a cell in state 2 is updated to state 3 as in Fig. 8(ii), according to transition rule 1. (2) Quiescent cells are updated to state 1 as in Fig. 8(iii), according to rule 2. Since the CA is asynchronous, the two cells to which this rule applies are not necessarily updated simultaneously. (3) A state-3 cell is updated to state 2 as in Fig. 8(iv), according to rule 13, thus extending the core of the signal



(a)



(b)

Fig. 11. (a) An Exit. Cells c_1 and c_2 may switch their states between 0 and 1 indefinitely and independently whenever transition rule 18 or 20 is applied; (b) exit operating on a signal. The signal can only be output from the Exit when both cells c_1 and c_2 revert to quiescent states like in (iii), otherwise it stays on the internal path of the Exit.

by one cell. (4) A state-2 cell is updated to state 1 as in Fig. 8(v), according to rules 10–12, thus initiating the withdrawal at the back of the signal. (5) State-1 cells are made quiescent as in Fig. 8(vi), according to rules 3–8, thus cleaning up garbage behind the signal.

Though due to the asynchronicity, a signal can have a wide variety of configurations during its transmission on a path [12], it is reliable even when garbage is left on the path by other signals, for example as in Fig. 9. Once a signal is produced on a path by a module, it will eventually reach its destination.

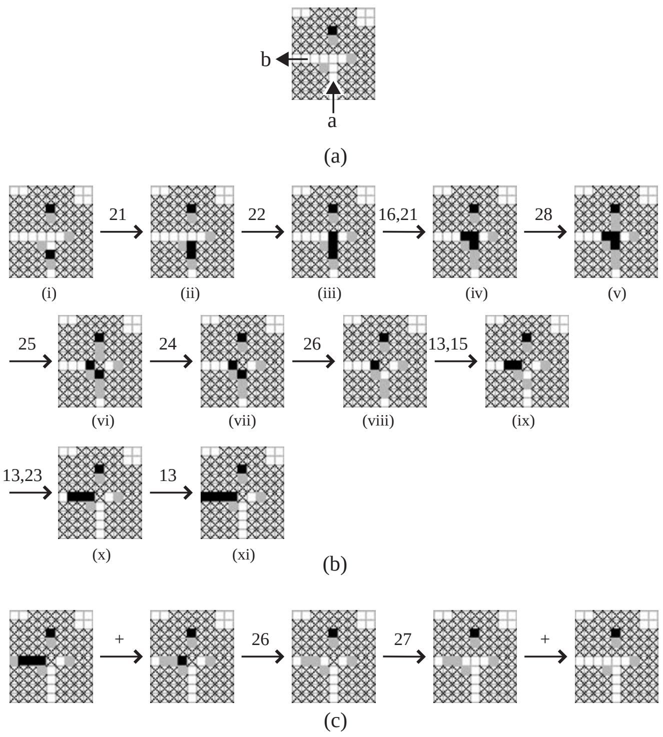


Fig. 12. (a) A Turn Core; (b) Turn Core operating on a signal arriving on its lower internal path, and (c) cleaning up the garbage after a signal at its right internal path has left the Exit to be attached to the Turn Core (see e.g. Fig. 20).

3.2. Configurations representing modules

In A_4, each primitive module in Fig. 4 can be further divided into several simple components: {Entrance, Exit, Turn Core, Multiplexer, Merge Core, Fork Core, Join}. We first describe the configurations of these components and their operations on signals.

3.2.1. Entrance

The Entrance in Fig. 10(a) is used to receive signals coming from the outside. The transition rules 13–17 in Fig. 6 are specified for an Entrance in A_4. Some of these rules are also required for the operation of other components.

A typical example of an Entrance operating on an input signal is given in Fig. 10(b). Due to the asynchronous nature of the CA, there are many different ways in which signals can be processed. To cope with the wide variety of signal configurations, the Entrance waits for a signal having a regular core in state 2 before accepting the signal, as in Fig. 10(b(iii)). The tail of a signal reduces to cells in state 1, as in Fig. 10(b(vii)), just outside the Entrance, and the tail will eventually be cleaned up once the signal enters the internal path of the Entrance.

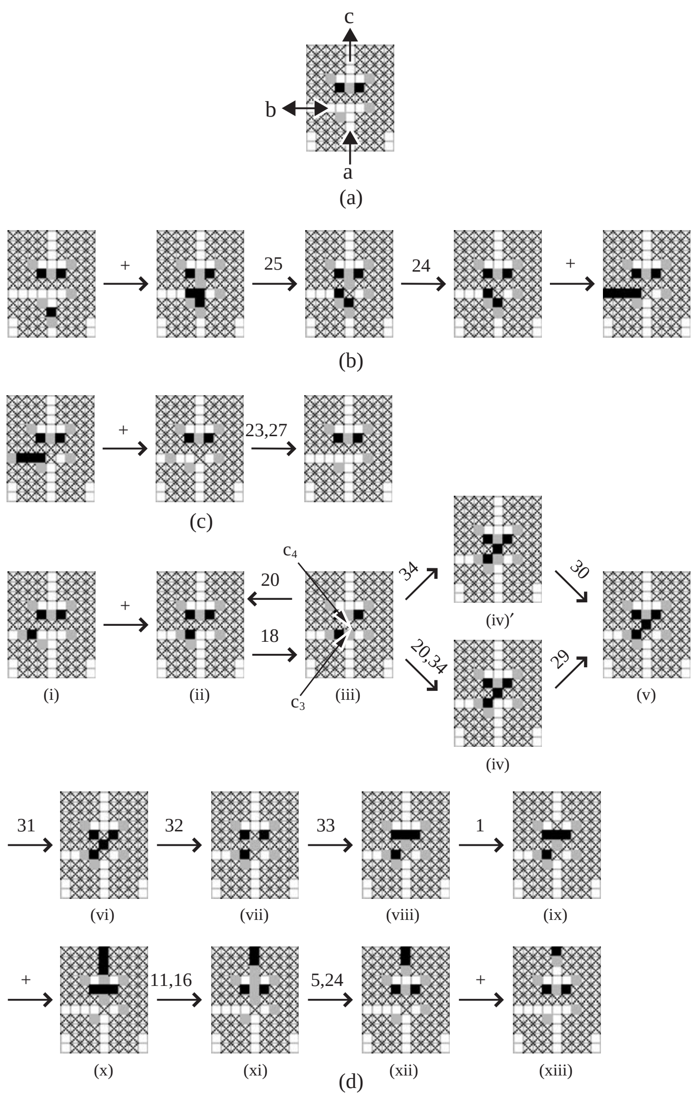


Fig. 13. (a) A Multiplexer; (b) multiplexer operating on a signal arriving on its lower internal path, and (c) cleaning up the garbage; (d) multiplexer operating on a signal arriving on its right internal path, in which cell c_3 in (iii) may switch its state between 0 and 1 until cell c_4 changes its state as in (iv) or (iv)′.

3.2.2. *Exit*

The Exit in Fig. 11(a) is used to transfer signals arriving on its internal path of quiescent cells to the outside path (see Fig. 11(b)). The transition rules 18–20 in Fig. 6 are required for the Exit.

3.2.3. *Turn Core*

The Turn Core in Fig. 12(a) is used to transfer signals arriving on its internal path *a* to internal path *b* as in Fig. 12(b). The transition rules 21–28 in Fig. 6 are required for the Turn Core.

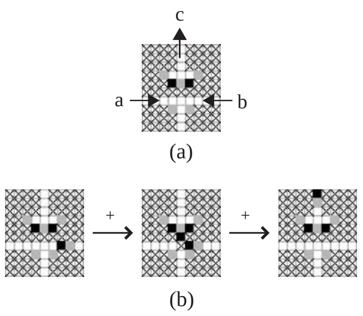


Fig. 14. (a) A Merge Core. (b) Merge Core operating on a signal on its right internal path.

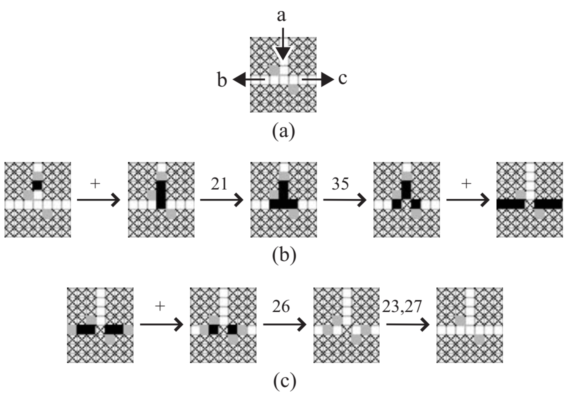


Fig. 15. (a) A Fork Core; (b) Fork Core operating on a signal, and (c) cleaning up the garbage after two signals have left the Exits to be attached to the Fork Core (see Fig. 18).

3.2.4. *Multiplexer*

The Multiplexer in Fig. 13(a) is used to transfer signals arriving on its internal path *a* to path *b*, working as a Turn Core (see Fig. 13(b)). Moreover, it transmits signals coming from internal path *b* to path *c* as in Fig. 13(d). The transition rules 29–34 in Fig. 6 are required for the Multiplexer.

3.2.5. *Merge Core*

The Merge Core in Fig. 14(a) is a slight modification of the Multiplexer in Fig. 13(a) which is used to redirect signals arriving on internal path *a* or *b* to path *c* (see Fig. 14(b)).

3.2.6. *Fork Core*

The Fork Core in Fig. 15(a) duplicates a signal arriving on its internal path *a* into two signals each on paths *b* and *c* (see Fig. 15(b)). The transition rule 35 in Fig. 6 is required for the Fork Core.

3.2.7. *Join*

Fig. 16(a) gives the JOIN, in which signals arriving on two internal paths $a(b)\langle c\rangle$ and $b(c)\langle a\rangle$ of the JOIN give rise to a single signal on path $c(a)\langle b\rangle$, for example as in Fig. 16(b). A single arrival on only

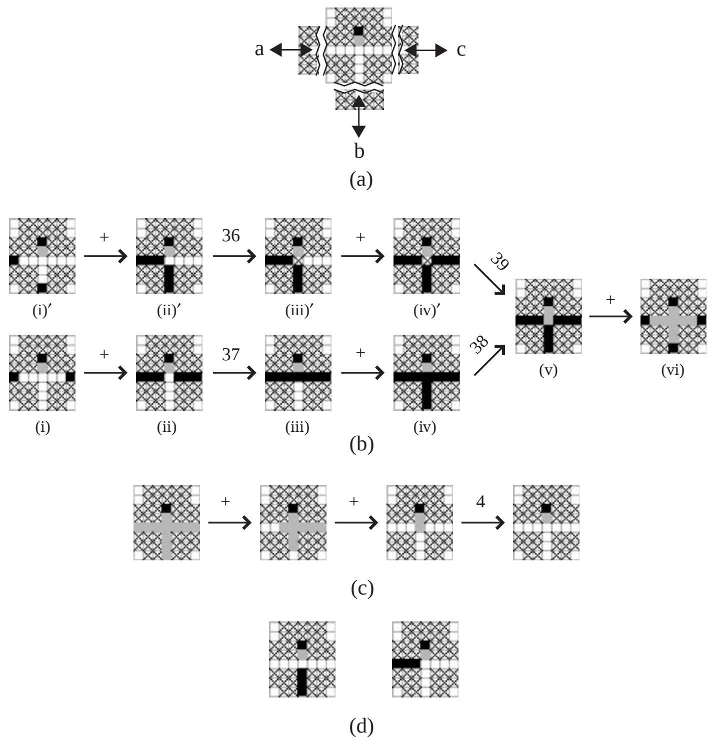


Fig. 16. (a) A Join. Internal paths a, b, c can be stretched freely; (b) join operating on a pair of signals arriving on its two internal paths, and (c) cleaning up the garbage after the three signals in (vi) have left the Multiplexers to be attached to the Join (see Fig. 19); (d) two pending signals.

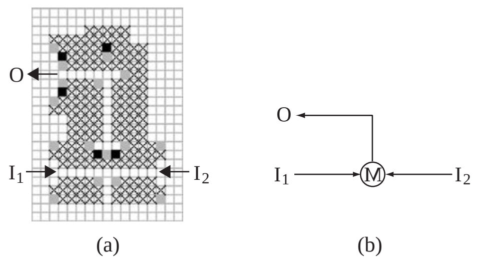


Fig. 17. (a) The configuration of the MERGE module, and (b) its alternative symbol.

one of the paths of the JOIN remains pending until another signal arrives from one of the other paths as in Fig. 16(d). The transition rules 36–39 in Fig. 6 are required for the JOIN.

Using all the components described above, we can assemble the modules in Fig. 4 in the cellular space of A_4. Fig. 17 gives the construction of the MERGE module by combining (or possibly overlapping) components {Entrance, Exit, Turn Core, Merge Core}. Fig. 18 gives the construction of the FORK module from components {Entrance, Exit, Fork Core}. Also, Fig. 19 gives the construction of the TRIA module from components {Entrance, Exit, Turn Core, Multiplexer, Join}, where the positions of the input

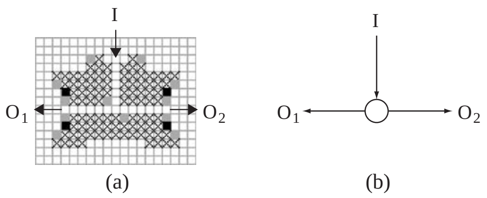


Fig. 18. (a) The configuration of the FORK module, and (b) its symbol.

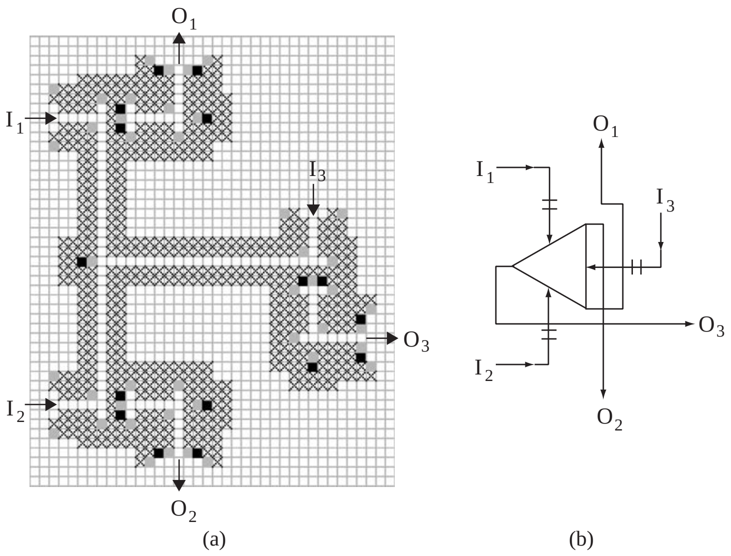


Fig. 19. (a) The configuration of the TRIA module, and (b) its alternative symbol.

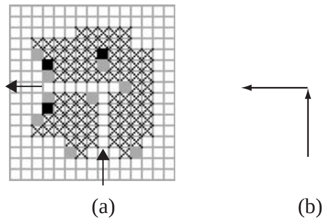


Fig. 20. (a) The configuration of the LR-Turn module, and (b) its symbol.

and output paths differ from those of the TRIA module in Fig. 4(c). Furthermore, a special module called LR-Turn module constructed from {Entrance, Exit, Turn Core} is given in Fig. 20, it can be used to change the directions of signals being transferred on the paths to the left or right.

The operations of the MERGE, FORK, TRIA, and LR-Turn modules embedded in A_4 must obey the operating conditions in Section 2. In particular, according to Condition 7, a signal is expected to be received by a module on one of its input paths, after the module completes the processing of all previous input signals except for those that are still pending on different paths. In this case, although the garbage

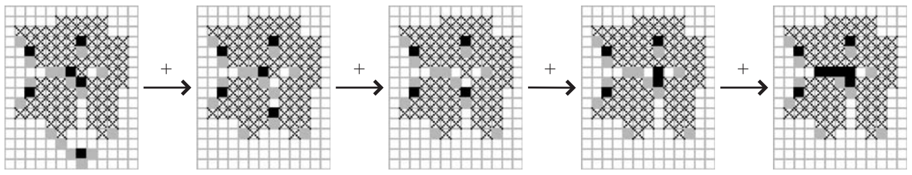


Fig. 21. Garbage inside the LR-Turn module delays the assimilation of the input signal as long as it remains, but cannot interfere with the signal.

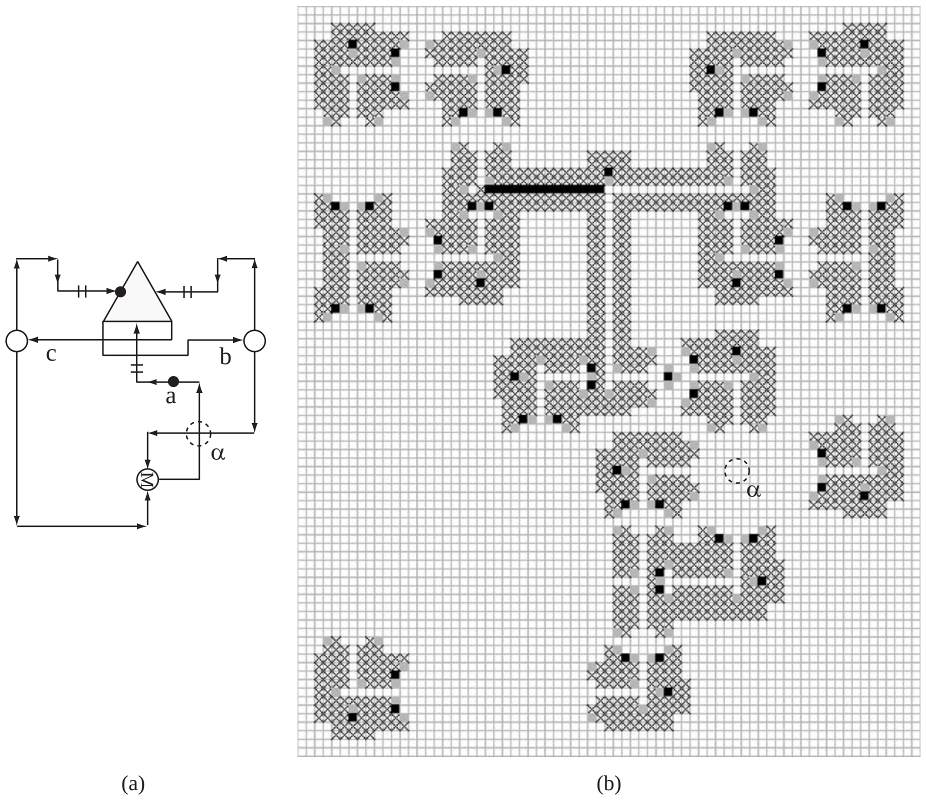


Fig. 22. (a) A delay-insensitive circuit scheme, and (b) its implementation in A_4. The dashed circle α indicates a cross point of two paths. A signal put initially on path *a* will run around in the circuit indefinitely. The TRIA module with a pending signal on its left input path toggles the signal on path *a* to path *b* or *c* in turn.

left during the processing may still remain when the module starts to process a new input signal, it never interferes with the module's correct operation of the new signal (for example, see Fig. 21).

3.3. *Laying out circuits to simulate Turing machines*

Assume a DI-circuit composed by a network of the primitive modules in Fig. 4, in which each primitive module is implemented in our asynchronous cellular automaton A_4 by a local configuration of cells, with

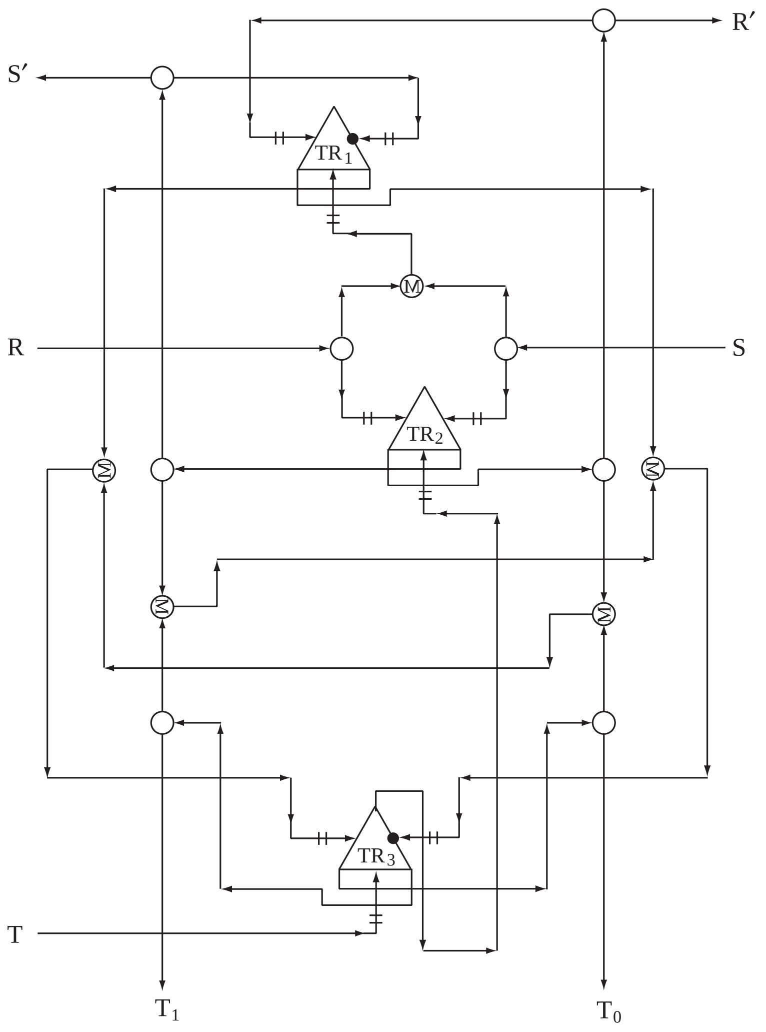


Fig. 23. The DI-circuit scheme [20] for constructing the S module. The two signals pending on the right input paths of TRIA modules TR_1 and TR_3, respectively, represent the S module being in state 1, whereas signals pending on the left input paths of TR_1 and TR_3, respectively, represent this circuit being in state 0.

their inputs and outputs connected to each other by paths of quiescent cells, over which signals are transferred from a source module to a destination module. Once a signal reaches a destination module, the module operates on it, which usually results in one or more signals on its output paths. Once a DI-circuit is formed, computing tasks can be carried out by inputting signals to appropriate paths of the circuit, giving rise to output signals on appropriate paths.

An example of a DI-circuit is given in Fig. 22, along with its configuration in A_4. Each primitive module in Fig. 22(b) is separated from the other modules by at least one quiescent cell, such that there is no interference between the modules at the end of any path when they receive input signals and produce output signals on the path.

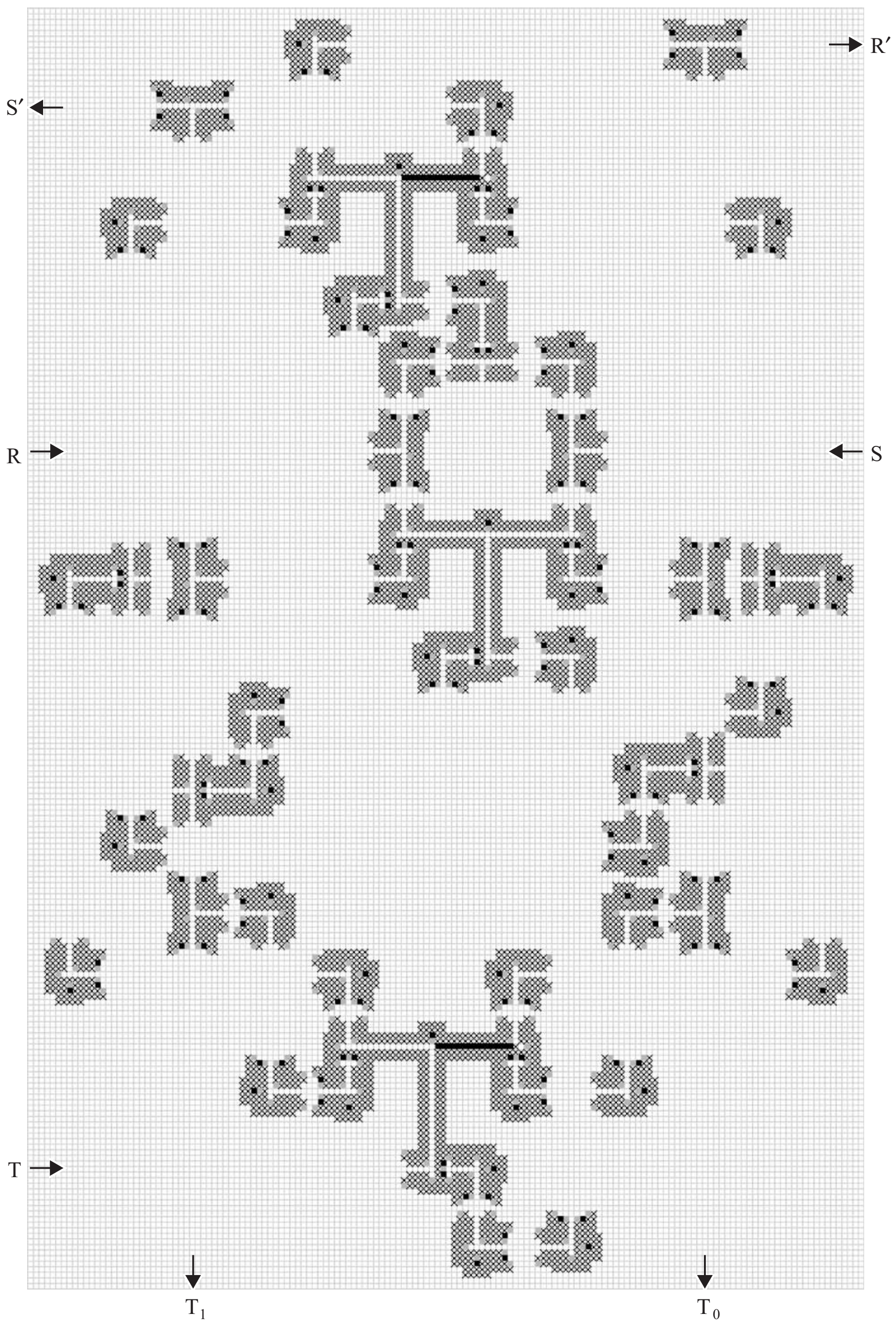


Fig. 24. The configuration of an S module in state 1.

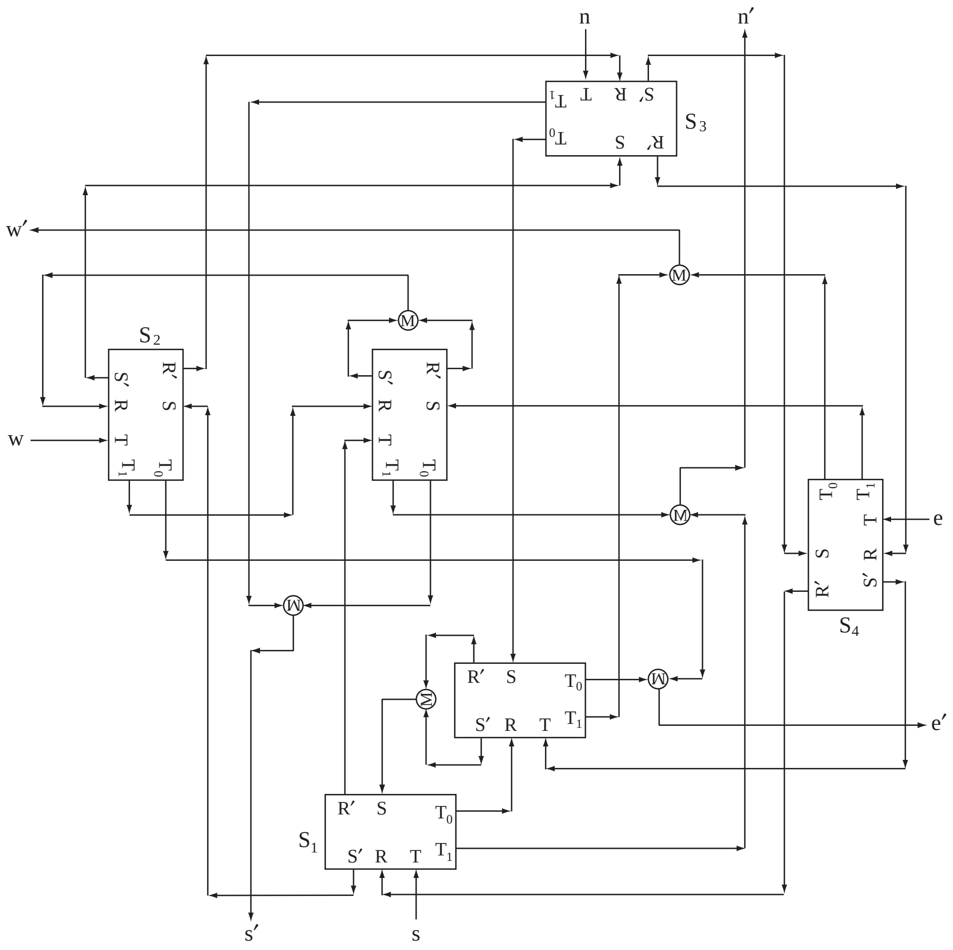


Fig. 25. The DI-circuit scheme for constructing the RE module, in which the four S modules S_1, S_2, S_3, and S_4 in states 1, 1, 1, and 1, respectively, represents the RE being in state 1, whereas S_1, S_2, S_3, and S_4 in states 0, 0, 0, and 0, respectively, represents the RE being in state 0.

This circuit is designed such as to avoid the collisions of signals on crossing paths: Two signals cannot arrive simultaneously at cross point α in Fig. 22. This is important, because two signal heads colliding at a cross point of paths would cause the deadlock of both of the signals, as A_4 has no transition rules to cope with such a situation, and lacks a third dimension via which crossings can be made (see also [12]).

We proceed by implementing the S module in Fig. 2 and the RE in Fig. 3 in our cellular space. The serial operation enables the S module and the RE to be constructed by using {MERGE, FORK, TRIA} according to Theorem 1. Fig. 23 gives the construction of the S module, and Fig. 24 gives its implementation in A_4. Moreover, following the construction in Fig. 25 along with the construction in Fig. 23, the RE module can be easily embedded in the cellular space of A_4. Because of its huge size, however, we do not show the configuration of the RE in this paper.

By implementing the RE module in Fig. 25, we can lay out a circuit of MERGE, FORK, TRIA, and LR-Turn modules in A_4 such as to construct an arbitrary Turing machine, as in [15]. Since at most one signal moves around between any REs in the circuit at any time, signals never collide on any path crossings in the network of modules, and hence, this circuit embedded in A_4 operates correctly as a simulated Turing machine. Consequently, the following theorem holds.

Theorem 2. *A_4 is computationally universal.*

4. Conclusion

In this paper, we presented a 4-state ACA with von Neumann neighborhood that uses rotation- and reflection-symmetric transition rules to describe the interactions between cells. This ACA model is capable of universal computation, in the sense that it allows a particular DI-circuit laid out in its cellular space to simulate a universal Turing machine. Since universality in computation is irrelevant to the efficiency of delay-insensitive computation, even DI-circuits operating in a strictly serial mode suffice to carry out universal computation [15]. We thus further reduced the number of cell states in the computation-universal ACA with von Neumann neighborhood as compared to the 5-state ACA model in [12].

We verified the correctness of our ACA model via a Java-based ACA simulator, by applying various asynchronous updating methods (see e.g. [22]) to iterate cells, for example, the stochastic scheme in [4] such that at every time step each cell assumes a certain probability to be updated. Moreover, since the design of signals is indispensable for laying out logic circuits in ACA models [1], witness the process of signal propagation in Fig. 8, the 4 states seem the minimal number of cell states required for an ACA with von Neumann neighborhood to be able to carry out universal computation. Further investigations, however, are needed to confirm this conjecture.

Acknowledgments

We are grateful to the anonymous referee for his/her helpful comments.

References

[1] S. Adachi, J. Lee, F. Peper, On signals in asynchronous cellular spaces, IEICE Trans. Inform. Systems E87-D (3) (2004) 657–668.
[2] S. Adachi, F. Peper, J. Lee, Computation by asynchronously updating cellular automata, J. Statist. Phys. 114 (1–2) (2004) 261–289.
[3] E.R. Banks, Universality in cellular automata, in: IEEE 11th Annual Symposium on Switching and Automata Theory, Santa Monica, CA, 1970, pp. 194–215.
[4] H.J. Blok, B. Bergersen, Synchronous versus asynchronous updating in the “game of life”, Phys. Rev. E 59 (1999) 3876–3879.
[5] M. Capcarrère, Cellular automata and other cellular systems: design & evolution, Ph. D. Thesis, Swiss Federal Institute of Technology, Lausanne, 2002.
[6] E.F. Codd, Cellular Automata, Academic Press, New York, 1968.
[7] U. Golze, (A-)synchronous (non-)deterministic cell spaces simulating each other, J. Comput. System Sci. 17 (1978) 176–193.

[8] U. Golze, L. Priese, Petri net implementations by a universal cell space, Inform. Control 53 (1982) 121–138.
[9] A. Hemmerling, On the computational equivalence of synchronous and asynchronous cellular spaces, J. Inform. Process. Cybern. 18 (7–8) (1982) 423–434.
[10] T.E. Ingerson, R.L. Buvel, Structures in asynchronous cellular automata, Physica D 10 (1984) 59–68.
[11] R.M. Keller, Towards a theory of universal speed-independent modules, IEEE Trans. Comput. C-23 (1) (1974) 21–33.
[12] J. Lee, S. Adachi, F. Peper, K. Morita, Embedding universal delay-insensitive circuits in asynchronous cellular spaces, Fund. Inform. 58 (3–4) (2003) 295–320.
[13] J. Lee, S. Adachi, F. Peper, K. Morita, Asynchronous game of life, Physica D 194 (2004) 369–384.
[14] J. Lee, F. Peper, S. Adachi, K. Morita, Universal delay-insensitive circuits with bi-directional and buffering lines, IEEE Trans. Comput. 53 (8) (2004) 1034–1046.
[15] K. Morita, A simple universal logic element and cellular automata for reversible computing, MCU 2001, Lecture Notes in Computer Science, vol. 2055, 2001, pp. 102–113.
[16] K. Nakamura, Synchronous to asynchronous transformation of polyautomata, J. Comput. System Sci. 23 (1981) 22–37.
[17] K. Nakamura, Asynchronous cellular automata and their computational ability, Systems Comput. Controls 5 (5) (1974) 58 –66.
[18] C.L. Nehaniv, Evolution in asynchronous cellular automata, in: R.H. Standish, M.A. Bedau, H.A. Abbass (Eds.), Artificial Life VIII, MIT Press, Cambridge, MA, 2003, p. 65.
[19] P. Patra, D.S. Fussell, Efficient building blocks for delay insensitive circuits, in: Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems, IEEE Computer Society Press, Silver Spring, MD, 1994, pp. 196–205.
[20] F. Peper, J. Lee, S. Adachi, S. Mashiko, Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers?, Nanotechnology 14 (4) (2003) 469–485.
[21] L. Priese, A note on asynchronous cellular automata, J. Comput. System Sci. 17 (1978) 237–252.
[22] B. Schönfisch, A. de Roos, Synchronous and asynchronous updating in cellular automata, BioSystems 51 (1999) 123–143.
[23] T. Serizawa, Three-state Neumann neighbor cellular automata capable of constructing self-reproducing machines, Systems Comput. Japan 18 (4) (1987) 33–40.
[24] T. Toffoli, Integration of the phase-difference relations in asynchronous sequential networks, in: G. Ausiello, C. Böhm (Eds.), Automata, Languages, and Programming, Lecture Notes in Computer Science, vol. 62, 1978, pp. 457–463.
[25] T. Toffoli, N. Margolus, Cellular Automata Machines, MIT Press, Cambridge, MA, 1987.
[26] Von Neumann, Theory of self-reproducing automata, in: A.W. Burks (Ed.), University of Illinois Press, 1966.
[27] S. Wolfram, Cellular Automata and Complexity, Addison-Wesley, MA, 1994.