The Fifteenth International Symposium on Artificial Life and Robotics 2010 (AROB 15th '10),
B-Con Plaza, Beppu,Oita, Japan, February 4-6, 2010

An Application of Self-Reproducing Loops to Defect-Tolerant Computation on Self-Timed Cellular Automaton

Tadashi Kunieda[1], Teijiro Isokawa[1*], Ferdinand Peper[2,1],
Ayumu Saitoh[1], Naotake Kamiura[1], Nobuyuki Matsui[1]

[1] Division of Computer Engineering, Graduate School of Engineering,
University of Hyogo, 2167 Shosha, Himeji, 671-2280, Japan.
[2] Nano ICT Group, National Institute of Information and Communications Technology, Japan
[*] Corresponding author, E-mail: **isokawa@eng.u-hyogo.ac.jp**

***Abstract*:** For the realization of nanocomputers it will be important to have built-in defect-tolerance, which is the ability to overcome the unreliability caused by defective components. This paper explores defect-tolerance for nanocomputers based on Self-Reproducing Loops, in which each of the loops in the system acts as a computational element, supporting the propagation of signals along transmission wires and their processing in logic elements. The loop-based design facilitates the adaptation to defects through the expansion of wires such as to prevent them from being blocked by defects. The proposed system is implemented on an asynchronously timed Cellular Automaton.

1 Introduction

Cellular Automata (CA) attract increasing attention as architectures for computers with nanometer-scale devices (nano-computers), because their regular structures and local connectivity offer much potential for manufacturing based on molecular self-assembly [1]. An obstacle to the realization of nanocomputers is the reduced reliability of nanometer-scale devices as compared to their VLSI counterparts, due to fabrication defects. Discarding chips with only a small amount of defects, however, will be inefficient for the imperfect manufacturing processes expected for molecular self-assembly.

For this reason, alternative approaches to defect-tolerance need consideration. There have been several studies on defect-tolerance in Self-Timed CA (STCA) [1], which is a type of asynchronous CA [2]. The resulting models, however, tend to suffer from a large overhead in terms of the number of transition rules required to implement defect-tolerance (95% of all rules), while computation itself requires only few rules.

This paper proposes an STCA model with defect-tolerant capability that is based on Self-Reproducing Loops (SRLs) [3]. Computation on the STCA is realized by embedding so-called *Brownian circuits* [4] on the cell space, whereby wires, crossovers of wires, and logic elements are implemented in terms of SRLs.

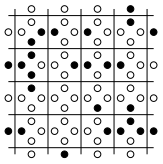


Fig. 1 Example of self-timed cellular space

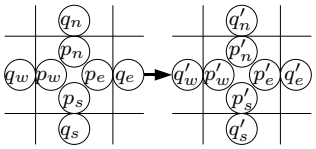
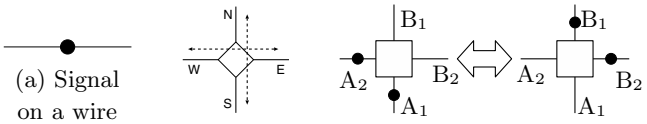


Fig. 2 Rule based on transition function f

2 Preliminaries

Self-Timed Cellular Automaton (STCA) [1] is a two-dimensional asynchronous CA of identical cells, each of which has a state that is partitioned into four parts in one-to-one correspondence with neighboring cells. If each partition of a cell state consists of 1 bit, the resulting 4 bits of the cell encode 16 states, and the cellular space is an array like in Fig. 1, where a filled circle denotes a 1-bit, and a open circle a 0-bit. Each cell undergoes transitions in accordance with a transition function f that operates on the four partitions p_n, p_e, p_s, p_w of the cell and on the nearest partition of each of its four neighbors q_n, q_e, q_s, q_w (Fig. 2), whereby a state symbol to which a prime is attached denotes the new state of a partition after update. Dummy transitions are not included in the transition function, so we assume that the left-hand side of Fig. 2 differs from the right-hand side. The transition rules of an STCA are rotation-symmetric, so each of the rules has four rotated analogues. In an STCA, transitions of the cells occur at random times, independent of each other. It is assumed that neighboring cells never undergo transitions simultaneously to prevent a situation in which such cells write different values in shared bits at the same time.

Computational elements needed in the model to ensure computational universality are [4]: a signal, a wire (possibly equipped with branching wires), a crossover of



(a) Signal on a wire (b) Crossover (c) Conservative Join

Fig. 3 Computational Elements

wires and a delay insensitive logic element, called Conservative Join(CJoin).

A signal can travel along a wire, like in Fig. 3(a). A crossover element (Fig. 3(b)) accepts a signal at port N (or S) and produces an output signal from port S (or N, resp.). In a similar way, the ports W and E are connected to each other.

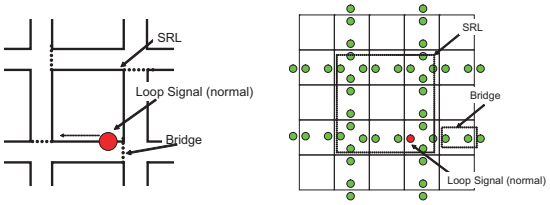
A CJoin is a two-input two-output logic element with four input/output wires A_1, A_2, B_1, B_2. Two signals from A_1 and A_2 (or B_1 and B_2 ,resp.) result in output signals from B_1 and B_2 (or A_1 and A_2, resp.) (see Fig. 3(c)) Both of the two input signals are necessary for the operation of a CJoin to take place.

Self-Reproducing Loops (SRLs) used in this paper work asynchronously and independently from each other; they are equipped with capabilities of shape-encoding and collision detection, like in [5]. Each SRL contains one "*loop signal*" that circulates clockwise in the loop. This signal is used for reproducing the loop, detecting defect neighboring loops, conducting computation, and expanding a transmission wire. The reproduction process starts when the loop signal arrives at a corner of the loop; here is where an arm starts to grow. An arm head will appear at the front of the arm, and if the cellular space in front of the arm head is not occupied by other loops, the loop signal will circulate in the loop to extract signals encoding the shape of the loop. These signals are processed at the arm head, resulting in a newly created loop structure. Due to asynchronously timing of the reproduction process in the loops, sometimes the arm head collides with other loops. In this case, the arm head withdraws itself and self-destructs, while the encoded signals remaining in the loop are deleted. The signal in the loop then starts again to scan the loop.

3 Implementing computational elements by Self-Reproducing Loops

3.1 Loops and Signals

The SRLs constructed in the model are connected to each other through a network structure in which individual SRLs exchange information via connections called *Bridges* (Fig. 4). Each loop has one signal circulating



(a) (b)

Fig. 4 (a) Loop connected to neighboring loops via bridges. (b) Implementation of Loop with Bridges on STCA.

clockwise in it, and when a signal arrives at a Bridge, the signal tries to interact with a signal in the neighboring loop at the other side of the Bridge. A signal in a loop can assume one out of a total of four states. At its default, a signal has state *Normal*, indicating that no information is present. To encode information in a signal we use the other three signal states, which we call *computational states*. The first of these is state *Comp0*, which is used as a general-purpose information carrier. The second and third of the computational states are *Comp1* and *Comp2*. Signals in these states are designed with the functionality of logic elements or crossovers of wires in mind. Signals in two loops interact with each other if one signal happens to be at one side of a Bridge and at the same time the other signal is at the other side. The basic interactions of signals in different states are as follows:

- If a signal in state *Normal* meets with a signal in state *Compi* ($i = 0, 1, 2$), then the states of the signals are exchanged.
- If two signals in state *Comp0* meet each other, then one signal changes its state to *Comp1* and the other to *Comp2*.
- If a signal in state *Comp1* meets with a signal in state *Comp2*, then both signals will change their states to *Comp0*.

3.2 Wires and Membranes

Loops form a homogeneous space at which signals interact with each other, but in order to conduct useful computations, a structure should be imposed on this space. Wires are formed on the space of SRLs through defining *sheath* cells, which separate the inside of wires from their outsides. Sheath cells assume one of three states $\{+, -, =\}$. In its basic form, a wire has $+$ cells at one side and $-$ cells at its other side (Fig 5). A wire is able to dynamically expand when it has insufficient capacity for its signals, for example due to the presence

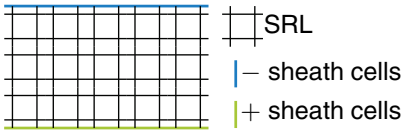


Fig. 5 Wire based on SRLs

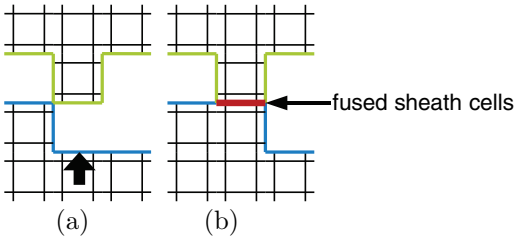


Fig. 6 Two wires getting in contact by expanding

of an SRL with a defect in it. Such a defect is detected through the failure of the SRL to circulate its signal through it. This failure will render the SRL incapable to react to external signals, which in turn will trigger the production of expansion signals emanating from the SRL. Expansion signals will push the sheath cells of the wire towards the outside, thus creating additional room for signals to pass through. Expansion may continue up to the point that the + sheath of one wire gets in contact with the − sheath of an adjacent wire, and in this case the + sheath and − sheath are fused at their contact areas, resulting in a sheath in state = (Fig. 6).

Another element imposing structure on a homogeneous space of SRLs is a so-called *membrane*. Signals in neighboring loops but separated by a membrane have interactions that are more complicated than the interactions described in section 3.1 between the signals in adjacent SRLs. The effects that membranes have on signals and their states appear as if the signals pass through the membranes, whereby the signals change their states. There are three types of membrane: *a*-membranes, *b*-membranes, and *c*-membranes. Fig. 7 shows the interactions of signals through membranes. An *a*-membrane is used to pass a *Comp0* signal in one direction when the signal at the other side of the membrane is in state *Normal* (see Fig. 7(a)). One possible use for an *a*-membrane is as a ratchet for *Comp0* signals. The other membranes also let signals pass through, and in the process change signal states in specific ways (Fig. 7(b) and 7(c)). However, these membranes differ from *a*-membranes by being reversible.

3.3 Logic Elements

Logic elements are constructed by combining membranes in appropriate ways. Fig. 8(a) shows the construction of a crossover element. When a *Comp0* signal

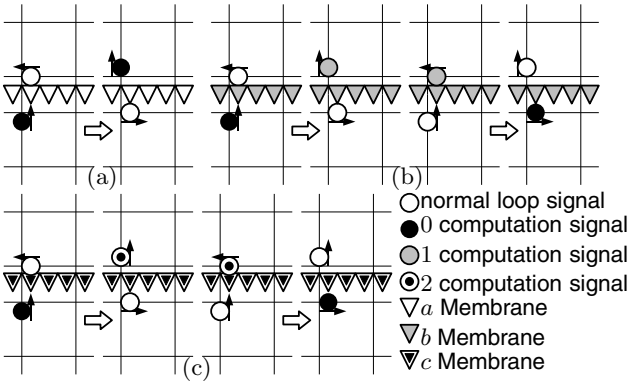


Fig. 7 Signal moving through membranes.

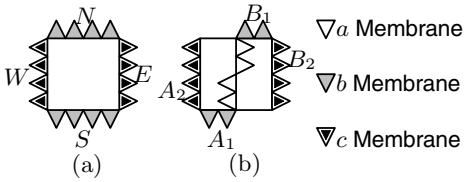


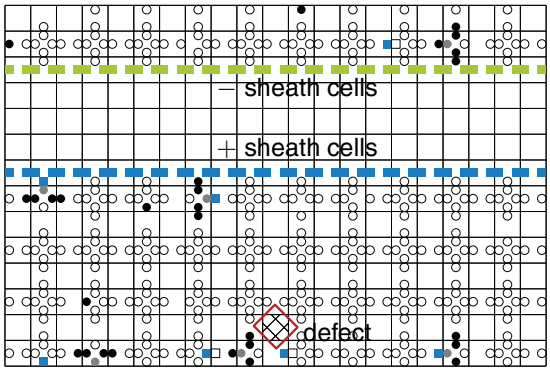
Fig. 8 Logic elements constructed by membranes. (a) Crossover and (b) Conservative Join

arrives at the port 'N', it passes through the *b* membrane and results in a *Comp1* signal inside the crossover element (see Fig. 7(b)). This signal will eventually pass through the membrane at the element's 'S' port, thereby changing its state back to *Comp0*. Passing a signal from port 'W' to 'E' or back works in the same way, except that the intermediate state of the signal inside the crossover element will be *Comp2*.

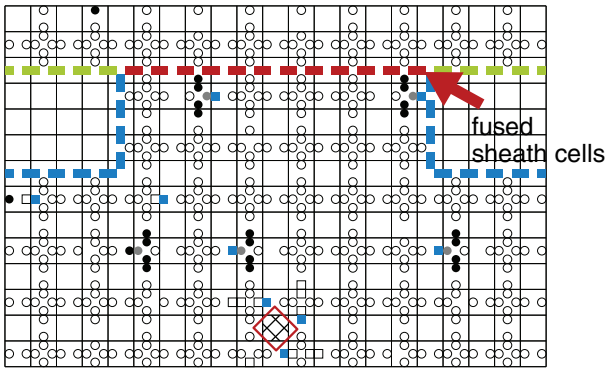
The construction of the Conservative Join is shown in Fig. 8(b). When each of the ports 'A_1' and 'A_2' receives a *Comp0* signal, the signals are transformed into a *Comp1* signal and a *Comp2* signal, respectively, after passing through the membranes of 'A_1' and 'A_2'. Once the signals are in the left chamber of the Conservative Join, they are unable to pass through the membrane in the center, unless they react with each other. As a result of the reaction of a *Comp1* signal with a *Comp2* signal, we get two *Comp0* signals in the left chamber, and only then these signals can pass through the center membrane to the right chamber. Once in the right chamber, the two *Comp0* signals may return to the left chamber via the center membrane. However, they also have the choice to stay in the right chamber and recombine into one *Comp1* signal and one *Comp2* signal, which can only leave the right chamber via the membranes of port 'B1' and port 'B2' respectiveley, thereby outputting one *Comp0* signal at each of these ports.

Table 1 Symbols encoding the states of cell partitions

0	1	2	3	4	5	6	7	8
	○	●	●	■	□	■	■	■



(a) Wire with defect in it at center bottom



(b) Wire after expansion primed by the defect

Fig. 9 Wire configuration on an STCA

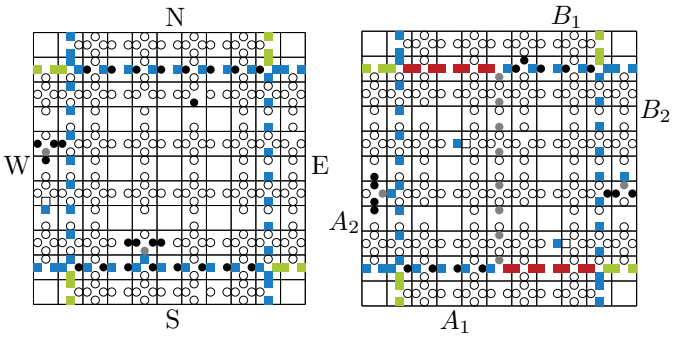
4 STCA implementations

Implementations of components described in the previous section are presented in this section. In the underlying STCA model, each cell partition can be in one of the 9 states indicated by the symbols shown in Table 1.

Fig. 9(a) shows two transmission wires implemented on the STCA, where the lower wire contains one defective SRL. As a result of expansion signals arriving at + sheath cells above the defect, the transmission wire expands up to the point that both wires get in contact with each other, after which the cells at the boundaries change their states to '=' (Fig. 9(b)). The STCA implementations of the crossover element and the Conservative Join are shown in Fig. 10.

5 Conclusions and Discussion

We have presented a computational model based on SRLs. The ability of SRLs to exchange signals in all directions fits well with the fluctuation-based nature of the underlying Brownian circuit model. The proposed model is robust to defects on account of the redundancy inherent in the use of multiple SRLs and in the expandability of wires. The proposed system is implemented on a 9^4 state-STCA with 362 transition rules. Simulations have shown the model to work correctly.



(a) Crossover (b) Conservative Join

Fig. 10 STCA implementations of logic elements

References

[1] F. Peper, J. Lee, S. Adachi, and S. Mashiko. Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers? *Nanotechnology*, 14:469–485, 2003.

[2] T. Kunieda, T. Isokawa, F. Peper, A. Saitoh, N. Kamiura, and N. Matsui. Reconfiguring Circuits Around Defects in Self-Timed Cellular Automata. In *Proc. of 8th International Conference on Cellular Automata for Research and Industry (ACRI2008)*, pages 200–209, 2008.

[3] C. G. Langton. Self-reproduction in cellular automata. *Physica D*, 10:135–144, 1984.

[4] J. Lee and F. Peper. On brownian cellular automata. In *Proc. of Automata 2008*, pages 278–291, UK, 2008. Luniver Press.

[5] Y. Takada, T. Isokawa, F. Peper, and N. Matsui. Asynchronous self-reproducing loops with arbitration capability. *Physica D*, 227:26–35, 2007.