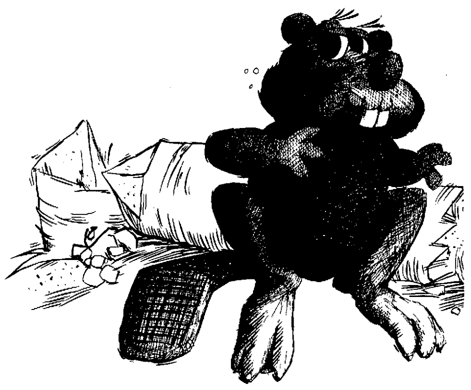
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT



Fault-Tolerant Schemes for Some Systolic Systems

Karel Culik II
Sheng Yu

CS-84-39

October, 1984

FAULT-TOLERANT SCHEMES FOR SOME SYSTOLIC SYSTEMS*

Karel Culik II & Sheng Yu

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

ABSTRACT

Fault tolerant schemes for several types of systolic systems are proposed. By fault tolerant scheme for certain type of systems, say unidirectional rings, we mean an algorithm that converts any given system of this type into an equivalent fault tolerant system of the same type. We consider a specific type of fault tolerance introduced by Kung and Lam. It is assumed that the faulty cells of a given system can be detected and 'switched' so that each of them passes its inputs to its outputs in specified order. To specify the degree of fault tolerance we introduce the notion of fault tolerance with respect to a collection Ω of subsets of the cells of a system. To be fault tolerant, the system must tolerate the failure of all the cells of any of the subset in Ω. We device fault tolerant schemes for trellis networks, one-way (unidirectional) cellular automata, one-way rings and two-way cellular automata and iterative arrays.

1. INTRODUCTION

The production of increasingly large and complex integrated circuits on a single chip leads to a decrease in integrated circuit yield, that is, the percentage of correct circuits. A strategy which can dramatically improve the yield is the design of "fault-tolerant" integrated circuits, that is circuits whose correctness does not require all constituent circuits to be correct. For more details see [11]. As in [11] we will study "fault-tolerant" systolic arrays. A systolic array is a system consisting of a large number of regularly interconnected synchronized

* This work was supported by the Natural Science and Engineering Research Council of Canada under Grant A-7403.

processors called cells. We will design systolic arrays which globally will function correctly even when some cells fail. By failure we mean that a cell just passes its inputs to its outputs (in certain order). Thus no cell really fails completely, rather we assume that each cell is designed so that it has this "bypassing" capability. We also assume that "faulty" cells can be detected and "switched" to bypass mode. The goal is to design a fault-tolerant system which will perform, essentially, the same computations as the original perfect system without faulty cells. Moreover we require it to do this with a minimal time delay. This is relatively easier to do for systems without feedback or with a simple feedback structure such as uni-directional, 2-dimensional arrays and rings considered in [11]. We study some additional types of systolic arrays including bi-directional linear arrays, that is, cellular automata which have rather complex feedback structure. Fault-tolerant cellular automata from a different point of view have been considered in [7] and [13]

We define a new notion of a systolic network N being fault tolerant with respect to Ω where Ω is a collection of subsets of the cells of N (for a fixed bypass function). Intuitively this means that N will perform, with no more than linear delay, essentially the same computations whenever the set of its faulty cells is in Ω. Obviously, we cannot ask that N performs the same computations if all its cells fail. It is also clear that it is easier to make N fault-tolerant with respect to a "small" rather than a "large" Ω.

We say that we have a fault-tolerant scheme for a given type of network if we give an algorithm which converts every network of this type into an equivalent fault-tolerant network of the same type. In Section 3 we show a fault-tolerant scheme for trellis networks [3] which are a generalization of trellis automata introduced in [4]. In order to be able to formalize our results we will use the notion of systolic network and simulation between networks as introduced in [3].

We illustrate our technique for the design of fault-tolerant trellis networks with a simple example. Figure 1.1.a shows a trellis network. Every cell performs the same function and it has no memory and every edge represents a unit delay (register), see Section 3 for details. If there is a faulty cell as in Figure 1.1.b, we can cut the network and reconnect it as shown in Figure 1.1.c. By the Cut Theorem of [11] the newly formed network is equivalent to the original one. But we can design a fault-tolerant system, i.e. a system which requires neither reconnections nor changes of the transition function for the nonfaulty cells. We only assume that the faulty cells are detected and made to "bypass" the data. The system will essentially create cuts automatically whenever there are faulty cells. It will work correctly no matter where the faulty cells are located. Also, it does not require any rearrangement of the input data.

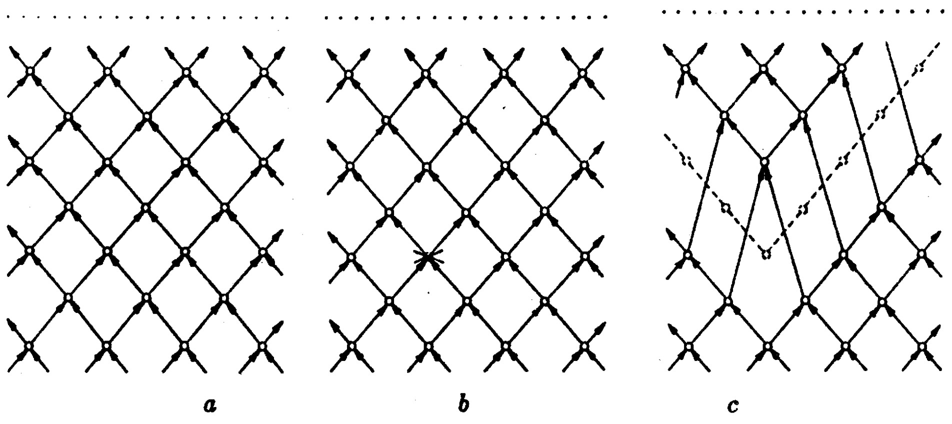


Figure 1.1

After giving the basic definitions in Section 2, we will describe the design of fault-tolerant trellis networks in Section 3. Then in Section 4 the fault-tolerant one-way cellular automata and one-way cellular rings are discussed. Finally in Section 5 we design fault-tolerant linear two-way cellular automata and iterative arrays.

In the first five sections we require that a fault tolerant scheme must convert a given systolic system of certain type into a fault tolerant system of the same type. In the last section we drop the requirement of the same type. We show that every linear bidirectional array (cellular automaton) or ring can be converted into a fault tolerant system of even a simpler type, namely a unidirectional systolic ring. We show this using the result of [3] that every bidirectional linear array or ring can be simulated by a twice slower unidirectional ring.

2. BASIC DEFINITIONS

Following [11] we consider fault-tolerant systolic systems which maintain the original data flow pattern and regular communication by bypassing faulty (defective) cells. In a faulty cell the normal computation (same for all cells) is no longer performed. Instead the data are simply passed from inputs to outputs in a fixed pattern and normal delay. Formally, we define a faulty cell (with m inputs and n outputs) as a special cell which performs the (bypass) function

$p(a_1, \ldots, a_m) = (a_{i_1}, \ldots, a_{i_n})$, where $m,n \geq 1$ $1 \leq i_1, \ldots, i_m \leq m$, i.e. every component of the output vector is a replica of a component of the input vector.

Our goal is to design systolic systems so that even with faulty cells they will perform the same computations or accept the same language as they do without faulty cells, possibly with some delay.

A necessary step in formalizing the notion of fault-tolerance is to make precise what it means that two systolic systems "perform the same computations". We will use the notion of *systolic network* and *simulation* between networks from [3]. Informally, we say that network N_1 simulates network N_2 if there is a mapping of the space-time diagram (unrolling) of N_2 to the space-time diagram of N_1 such that every computation on (unrolling of) N_2 can be recovered from the corresponding computation on (unrolling of) N_1. Intuitively, N_1 is said to (m,k)-simulate N_2 if any cell of N_1 corresponds to at most m cells of N_2 and if every cell of N_1 takes at most k time units to simulate one computation step of N_2.

In the above definition of (m,k)-simulation, k is locally constrained, i.e., every step of computation should be simulated by no more than k steps. We call it local (m,k)-simulation. If any n steps of computation started from the beginning can be simulated by no more than kn steps, then we call it global (m,k)-simulation. Notice that local (m,k)-simulation implies global (m,k)-simulation, and they both imply general simulation. But this is not in general true vice versa, except (1,1)-simulation which is the same being either local or global. Notice that (1,1)-simulation is a transitive and reflexive relation. If N_1 (1,1)-simulates N_2 and N_2 (1,1)-simulates N_1, then we say that N_1 and N_2 are equivalent. Clearly, mutual (1,1)-simulation is an equivalence relation.

In this paper, if there exists an m and k, $m,k > 0$, such that N_1 (1,1)-simulates N_2, we will simply say that N_1 simulates N_2. Obviously, simulation implies linear delay by this definition. A systolic network (system) without any faulty cell is called a perfect network (system).

Let A be a systolic network (system) with the set of cells Q and let Ω be a collection of subsets of Q, i.e. $\Omega \subseteq 2^Q$. We assume that a bypass function p is fixed. Let A_S be the systolic network obtained from A by replacing a subset S of its cells by faulty cells. If for each $S \in \Omega$, A_S simulates A, then A is called *fault-tolerant with respect to Ω*. When Ω is the set of all finite subsets of Q, we simply say A *is fault-tolerant*.

Notice that in the above definition all the cells of A_S except those in S are unchanged, i.e., they still perform the same transition function. And generally the pattern and timing of the input string should not be changed either.

Informally speaking, an (infinite) systolic network is fault-tolerant if it performs, with no more than linear delay, essentially the same computations whenever any finite number of its cells are faulty.

3. FAULT-TOLERANT TRELLISES

Since the trellis networks, trellises for short, are the basic tools to facilitate the study of iterative arrays, one-way systolic rings, cellular automata etc. and they themselves are a useful model too, we will start with the design of a fault-tolerant scheme for trellises.

Trellises have been introduced in [4], and studied in [5], [2], [9]. We use a definition which is similar to that in [9]. A trellis network is a system consisting of an infinite planar array of identical cells with unit time propagation delay between the cells. Each cell has two inputs, called left and right input, and two outputs, left and right output. Each output is a function of the two inputs. An input to a trellis is a string $w = a_1 \dots a_n$ where $a_1, \dots, a_n$ are the left inputs to n consecutive cells of the same row of the planar array. See Figure 3.1. The formal description follows:

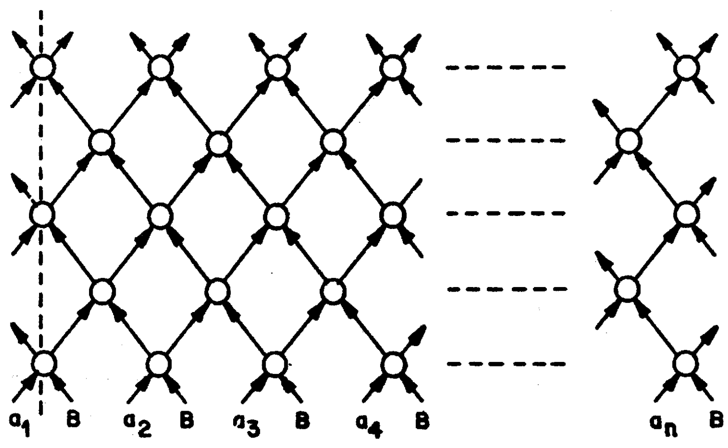


Figure 3.1

A *trellis network* is a system $A = (\Gamma, \Sigma, f)$ where Γ is the working alphabet with a special quiescent symbol $\# \in \Gamma$; $\Sigma \subseteq \Gamma$ is the input alphabet with blank symbol $B \in \Sigma$; $f : \Gamma^2 \to \Gamma^2$ is the transition function for every cell with the property that $f(B,B) = (\#,\#)$ and $f(\#,\#) = (\#,\#)$.

A trellis network with a subset of states Δ specified as accepting states is called a trellis automaton. It defines a language in the following way. A string $w = a_1 \ldots a_n$ is accepted by a trellis automaton if the left output of a cell in the column where a_1 enters is in an accepting set $\Delta \subseteq \Gamma$. A language is said to be accepted by A if it is exactly the set of all strings accepted by A.

Let us assume that ¢, \$ are in Γ and that every input string starts with ¢ and ends with \$. A trellis network is called bounded if its transition function f satisfies the property that $f(¢,a) = (¢,\#)$, $f(a,¢) = (\#,¢)$ and $f(\$,B) = (\#,¢)$, for any $a \in \Gamma$. Intuitively, a bounded trellis has only finitely many columns of cells. These columns are the ones where input symbols are read and the ones between them. The inputs in the remaining columns are always 'B's. The bounded trellises can be considered as the unrolling of the networks in Figure 3.2 or Figure 3.3. We will see later that the network in Figure 3.3 can be fault-tolerant but the network in Figure 3.2 cannot.

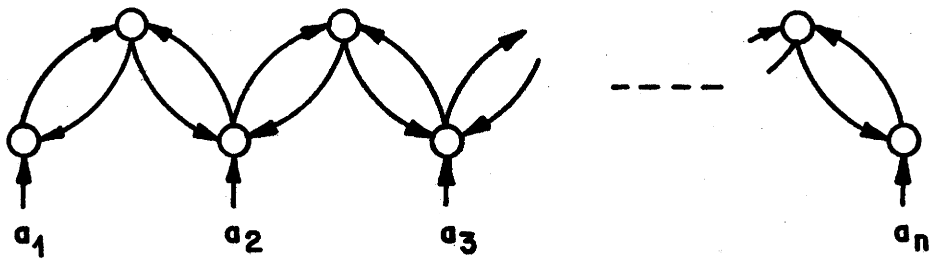


Figure 3.2

We now introduce a fault-tolerance scheme for trellises. A direction bit is added to every working symbol. The normal left and right input to a cell should have direction bits "$\to$" and "$\to$", respectively, and the cell produces a left output with "$\leftarrow$" and right output with "$\to$". Inputs with wrong direction bits to a cell result in the simple bypass of the input symbols (i.e., $f(a,b) = (a,b)$ if a or b has wrong direction bit). An example is shown in Figure 3.4. A computation on

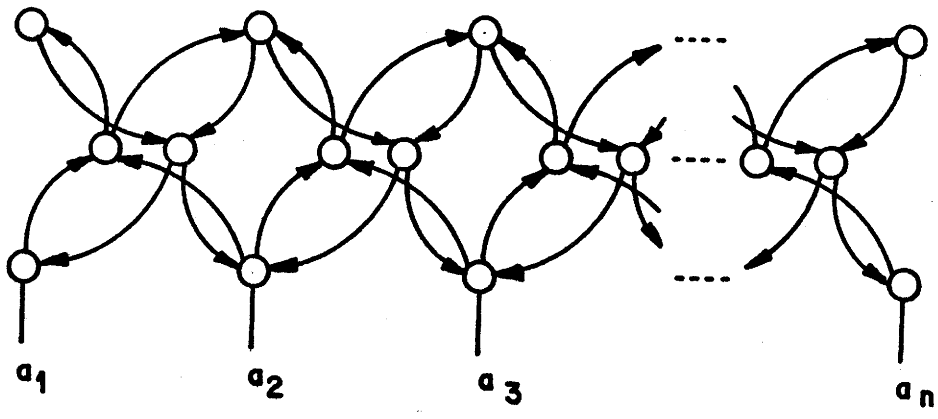


Figure 3.3

a perfect trellis is shown in Figure 3.4.a. The direction bits are omitted here. Figure 3.4.b shows how the same computation is simulated when the cell marked by "X" is faulty.

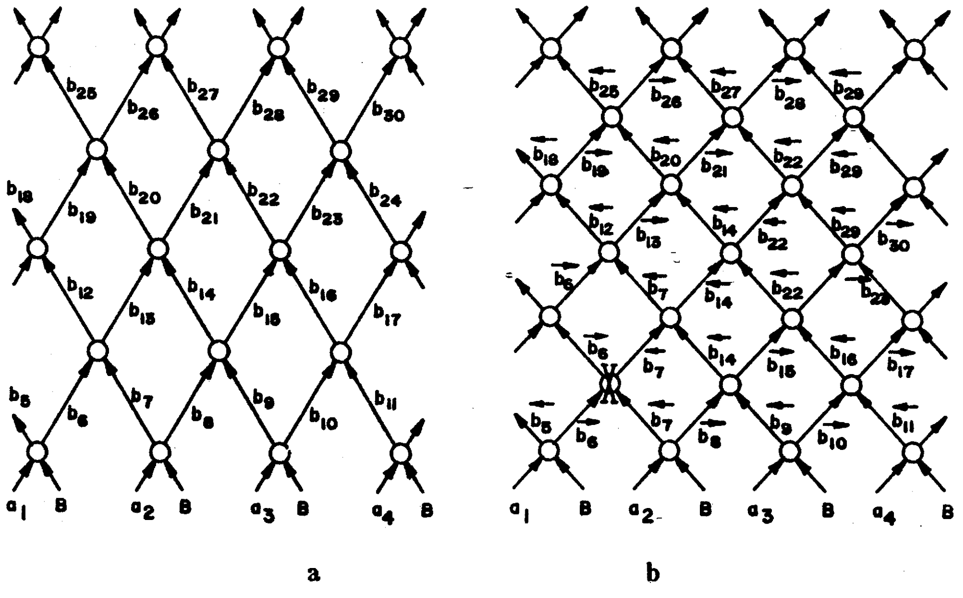


Figure 3.4

Theorem 1 *For any trellis network* $A = (\Gamma, \Sigma, f)$ *there exists an equivalent fault-tolerant trellis network* $A' = (\Gamma', \Sigma, f')$ *with bypass function* $p(a,b) = (a,b)$.

Proof: There are three steps in this proof:

(a) to construct A',

(b) to prove A' is equivalent to A,

(c) to prove A' is fault-tolerant.

Without loss of generality, we assume that

$$\Sigma \cap dom\ f = \varnothing.$$

A' is constructed as follows:

$$\Gamma' = \{\overleftarrow{a}, \overrightarrow{a} \mid a \in \Gamma - \Sigma\} \cup \Sigma;$$

$$f'(\alpha,\beta) = \begin{cases} (\overleftarrow{c}, \overrightarrow{d}), & \text{if } \alpha = a \in \Sigma, \quad \beta = B \text{ and } f(a,B) = (c,d), \\ & \text{or } \alpha = \overrightarrow{a}, \qquad \beta = \overleftarrow{b} \text{ and } f(a,b) = (c,d) \\ (\alpha,\beta), & \text{otherwise.} \end{cases}$$

If A is bounded, then

$$f'(¢,\alpha) = \begin{cases} (¢,\#), \text{ if } \alpha = \overleftarrow{a} \in \Gamma'; \\ (¢,\alpha), \text{ if } \alpha \in \Sigma \text{ or } \alpha = \overrightarrow{a} \in \Gamma'; \end{cases}$$

$$f'(\alpha, \text{¢}) = \begin{cases} (\#, \text{¢}), \text{ if } \alpha = \bar{a} \in \Gamma'; \\ (\alpha, \text{¢}), \text{ if } \alpha = \bar{a} \in \Gamma. \end{cases}$$

To prove (b), it suffices to show that for any input word, a cell u_{ij} of A has input (a,b) and output (c,d) if and only if the cell v_{ij} of A' has input $(\bar{a},\bar{b})$ $((a,b)$, if $a \in \Sigma$ and $b = B)$ and output $(\bar{a},\bar{d})$, where u_{ij} and v_{ij} are the cells in ith row and jth column of A and A', respectively. This can be easily proved by induction on the number of rows. We leave this part of the proof to the reader.

Let Q and Q' be the set of cells in A and A' respectively. And let S be an arbitrary finite set of Q'. We now prove that there exists a k such that A's $(1,k)$ simulates A for any input string w $(|w| = n)$.

We first define a 1-1 mapping $\rho : Q \to Q'_s$ where Q'_s is obtained from Q' by replacing every cell of S by a faulty cell. ρ is defined recursively.

(i) $\rho(u_{1,j}) = v_{k,j}$, $k = \min\{t \mid t \geq 0 \text{ and } v_{t,j} \notin S\}$, for $j = 0, 2, \ldots, 2n$.

(ii) $\rho(u_{i,j}) = v_{k,j}$, $k = \min\{t \mid t > l, t > m, v_{t,j} \notin S\}$ where $v_{l,j-1} = \rho(u_{i-1,j-1})$ (or $l = 0$ if $j = 1$) and $v_{m,j+1} = \rho(u_{i-1,j+1})$ (or $m = 0$ if $j = 2n$).

Claim 1 Let $v_{i,j}$ be cell of A'_s and $(\bar{a},\bar{b})$ (or ((a,B) if $a \in \Sigma$) be the pair of inputs to $v_{i,j}$. Let $k = \min(t \mid t \geq i \text{ and } v_{t,j} \notin S\}$. Then $v_{l,j}$ will have input $(\bar{a},\bar{b})$ and output $(\bar{a},\bar{b})$ for all $i \leq l < k$. And $v_{k,j}$ will have input $(\bar{a},\bar{b})$ and output $(\bar{c},\bar{d})$ if $f(a,b) = (c,d)$ and undefined otherwise.

Intuitively, Claim 1 says that any pair of inputs to a cell of A'_s will eventually be passed to the first nonfaulty cell of the same column starting from that cell and this cell will perform the computation which was supposed to be done by the faulty cell.

If $v_{i,j}$ is not faulty itself, then Claim 1 is true trivially. If $v_{i,j}$ fails, both

$\vec{a}(a)$ and $\vec{b}(B)$ will be bounced back by the two neighboring columns because the orientation of the direction bits. By the definition of f', $\vec{a}(a)$ and $\vec{b}(B)$ will be passed forth and back to every next cell of the same column until the first non-faulty cell $v_{k,j}$ which will do the supposed computation. See Figure 3.5.

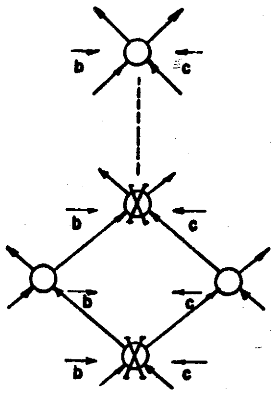


Figure 3.5

Claim 2 $u_{i,j}$ has input (a,b) and output (c,d) if and only if $\rho(u_{i,j})$ has input $(\vec{a},\vec{b})$ ((a,b) if $a \in \Sigma$ and $b = B$) and output $(\vec{c},\vec{d})$.

We prove the claim by induction on the number of rows of A. It holds for $i = 1$ by simply applying Claim 1 and the definition of ρ. Assume Claim 2 is true for all $i \leq n$. Consider $i = n+1$. Let $u_{n+1,j}$ has input (a, b). Then a is the right output of $u_{n,j-1}$ and b is the left output of $u_{n,j+1}$. Therefore $\vec{a}$ and $\vec{b}$ should be the right and left output of $\rho(u_{n,j-1})$ and $\rho(u_{n,j+1})$, respectively. Assume $v_{p,j-1} = \rho(u_{n,j-1})$, $v_{q,j+1} = \rho(u_{n,j+1})$. Let $k=\min\{t \mid t > p, t > q, v_{t,j} \notin S\}$. By the definition of ρ, $v_{k,j} = \rho(u_{n+1,j})$. By Claim 1, $v_{k,j}$ has input $(\vec{a}, \vec{b})$ and output $(\vec{c}, \vec{d})$ if $u_{n+1,j}$ has input (a, b) and output (c,d)./p c output (c,d). Let ρ has input $(\vec{a},\vec{b})$ and output $(\vec{c},\vec{d})$. Assume $u_{n+1,j}$ has input (a',b'), and $a' \neq a$ or $b' \neq b$. Then $\rho(u_{n+1,j})$ would have input $(\vec{a}',\vec{b}')$, $\vec{a}' \neq \vec{a}$ or $\vec{b}' \neq \vec{b}$ by the above proof and the definition of A'. This is a contradiction. So, $u_{n+1,j}$ must have input (a,b). The fact that

$u_{n+1,j}$ should have output (c,d) follows by the definition of f'. That completes the proof of Claim 2.

To complete the proof of Theorem 1 we define mapping $I_1: Q' \rightarrow IN$ by $I_1(v_{i,j}) = i$, for $v_{i,j} \in Q'$. Let

$$k = \max\{I_1(\rho(u_{i,j}))-I_1(\rho(u_{i-1,j})) \mid u_{i,j},u_{i-1,j} \in Q\}.$$

Since S is finite, k exists. Therefore A'_s $(1,k)$-simulates A, and A' is fault-tolerant. □

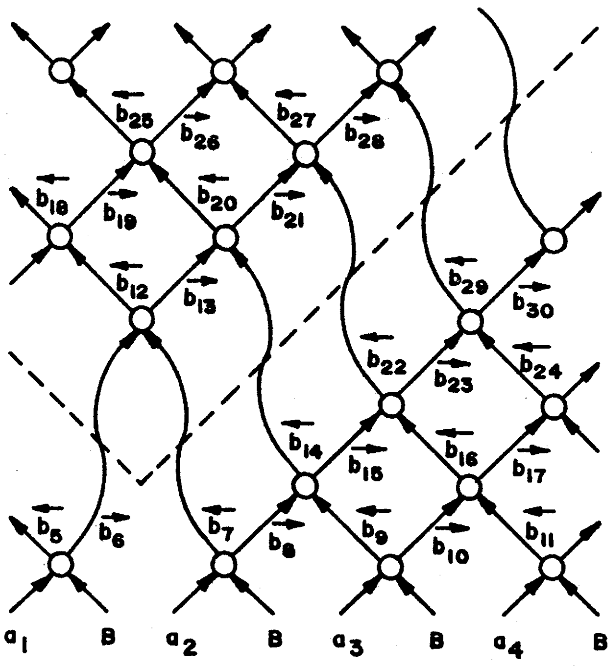


Figure 3.6

Figure 3.6 is a modified redrawing of Figure 3.4.b. From this diagram, we can see that our fault-tolerant scheme for trellises actually creates a cut or cuts when one or more cell fails. Two units of delay are added to every edge of a cut. Figure 3.7 indicates the cuts formed when two failed cells are in different mutual positions. Each cut divides the trellis into two parts, source and destination, and the source includes all the cells which have external inputs. The cuts are formed not physically, but rather logically by the transition function because the existence of the faulty cells.

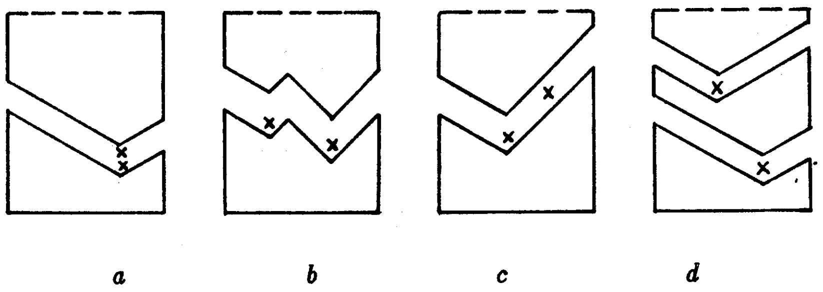


Figure 3.7

In the next theorem, we will see that a trellis formed by the above scheme is fault-tolerant with respect to a set Ω which properly contains the set of all finite subsets of Q.

For a trellis network $A = (\Gamma, \Sigma, f)$ and an integer $K > 0$, let Ω_k be the collection of all the subsets of Γ which do not contain k adjacent cells from any single column.

Theorem 2 *Given a trellis network $A = (\Gamma, \Sigma, f)$ and integer $k > 0$, we can construct an equivalent trellis network A' which is fault tolerant with respect to Ω_k.*

The construction of A' is the same as in the proof of Theorem 1. It is easy to prove by induction on the number of rows in A that any n rows of A can be simulated by no more than kn rows of A'. Hence, A'_S globally $(1,k)$-simulates A for any $S \in \Omega_k$ and thus A' is fault tolerant. Now, we will consider how the distribution of the faulty cells affects the delay in the simulation in Theorem 1.

Let S be the set of faulty cells. To calculate the delay caused by the faulty cells of S, we define a relation "$<$" between those cells in S. Let v_{ij} be a cell on the ith row and jth column of a trellis. Let $v_{i_1j_1} < v_{i_2j_2}$ for $v_{i_1j_1}, v_{i_2j_2} \in S$, if and only if $i_2 - i_1 > | j_2 - j_1 |$, i.e., they form two separate cuts or one cut of two cells thick (see Figure 3.7). Adding two pseudo faulty cells $v_{-\infty}$ and v_{∞} such that $v_{-\infty} < v < v_{\infty}$ for any cell $v \in S$, all the faulty cells of S involved in a computation form a lattice with respect to the relation "$<$". We call this lattice the lattice of the faulty cells.

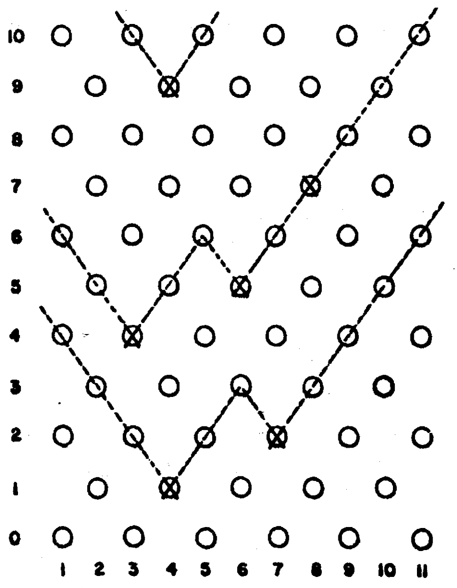
Theorem 3 *Let A' be a fault-tolerant trellis network constructed according Theorem 1 from a trellis network A, and S be a set of cells in A'. Then the delay of a computation by A'_S in comparison with A is no more than* $2(l-1)$ *where l is the length of the longest path in the lattice of the faulty cells.* □

Instead of giving a detailed constructive proof, we show the construction on an example, which explains the essential idea of the proof. Figure 3.8.a shows a trellis with failed cells (only the ones involved in the computation are shown in the diagram). Figure 3.8.b shows the faulty-cell lattice of the computation. From the faulty cell lattice we predict the cuts formed in the computation by the following rules.

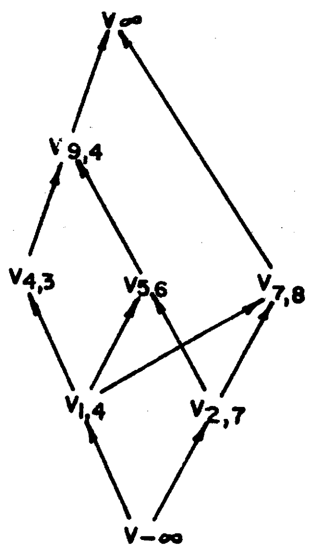
(1) All the cells directly above $v_{-\infty}$ form a cut.

(2) All the cells directly above a cell of a cut form a cut.

(3) Repeat (2) and stop at v_{in}.



a



b

Figure 3.8

By u being directly above v we mean that $v < u$ and there is no w such that $v < w < u$.

Corollary 1 *Let a set S in Ω contains a whole column of a trellis network N. Then N is not fault-tolerant with respect to Ω.*

This is true since the faulty cells of the whole column will form a infinite path in the faulty-cell lattice.

Corollary 2 *Let a trellis network N be fault-tolerant with respect to Ω and $S \in \Omega$ be a set of cells of a whole diagonal line of N. Then the delay of the computation of A'_S is no more than two time units.*

No two cells of a diagonal line are in the relation "$<$". Thus, every path from v_∞ to $v_{-\infty}$ is of length 2.

The unrollings of both the networks in Figure 3.2 and Figure 3.3 are trellises. By Corollary 1, the network in Figure 3.2 is not fault-tolerant. The network in Figure 3.3 uses twin cells to simulate one cell of the above network. This network can be fault-tolerant if there is no pair of twin cells with both its cells faulty.

The results on trellises are easily extended to some other regular systolic systems as shown in the next section.

4. FAULT-TOLERANT ONE-WAY CELLULAR AUTOMATA AND ONE-WAY CELLULAR RINGS

A one-way cellular automaton (OCA) is a system shown in Figure 4.1.a. Every cell of the system is a finite state machine with an input and an output. The output is the function of the current state and the input. The system has parallel input, i.e., the input symbols initiate the same number of adjacent cells at the same time. An OCA can also be considered as a synchronous network depicted in Figure 4.1.b each cell of which is simply a function and each edge is an unit delay. (See [2]). Formally, an OCA is a quintuple $A = (Q, \Sigma, \Gamma, f, \Delta)$ where Q is the state set, Σ is the initial input alphabet, Γ is the output and input alphabet, $f : \Sigma \cup \Gamma \times Q \rightarrow Q \times \Gamma$ is the transition function, and $\Delta \subseteq \Gamma$ is the set of accepting symbols.

The space-time diagram of a systolic network N is formalized in [3] as the unrolling of N. The unrolling of an OCA is a trellis. Therefore, the result of the

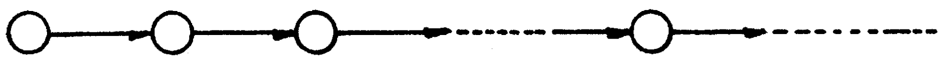


Figure 4.1.a

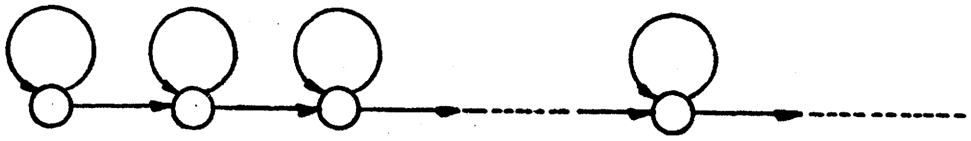
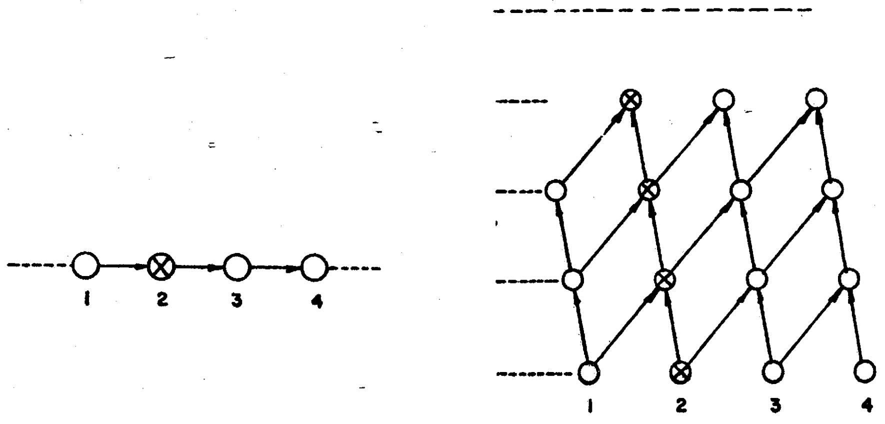


Figure 4.1.b

last section can be easily used for *OCA*'s. Let T be the unrolling of an *OCA* A. If there is a faulty cell in A (see Figure 4.2.a) then there is a whole failed diagonal line in T (see Figure 4.2.b). Assume that $B \in Q$ is the pre-initial state for every cell of the *OCA*. We have the following theorem.



a b

Figure 4.2

Theorem 4 *Let $A = (Q,\Sigma,\Gamma,f,D)$ be an OCA. Then there exists a fault-tolerant OCA A' with the bypass function $p(a,b) = (a,b)$ such that*

(1) $L(A') = L(A)$;

(2) *the delay by k failed cells of A' is $2k$;*

(3) *k more cells are used by A' than A if k is the number of the failed cells involved in the computation.* □

The construction of $A' = (Q',\Sigma,\Gamma',f',\Delta)$ uses the same principle as that of Theorem 1. However, it looks simpler since the faulty-cell pattern is more regular in a trellis which is the unrolling of an *OCA*. Let

$$Q' = Q \cup \Gamma$$

$$\Gamma' = \Gamma \cup \{\bar{q} \mid q \in Q\} \cup$$

$$f'(\alpha,p) = \begin{cases} f(a,p), & \text{if } \alpha = a \in \Sigma \cup \Gamma, p \in Q; \\ (q,\bar{p}), & \text{if } \alpha = \bar{q} \in \Gamma', p \in Q; \\ (B,\bar{p}), & \text{if } \alpha = B, p \in Q. \end{cases}$$

Properties (1) and (2) follow directly by Theorem 1 and Corollary 2, respectively. Property (3) holds since the time delay means that more diagonal lines are involved.

Intuitively, the bypass function $p(a,b) = (a,b)$ simply means an extra unit delay. By this scheme, an *OCA* will work correctly by just replacing the faulty cells by simple registers, see Figure 4.3.

This scheme is explained in an example in Figure 4.4, where the unrollings of both the perfect and faulty networks are shown. The dotted lines show where the cut is.

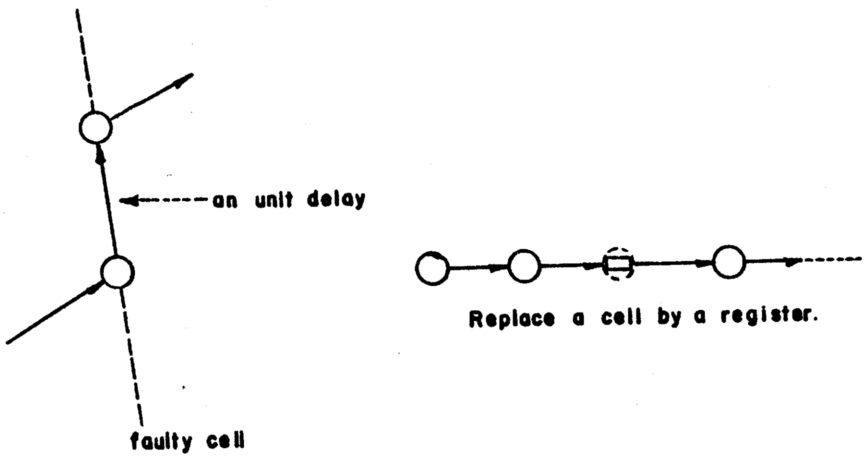


Figure 4.3

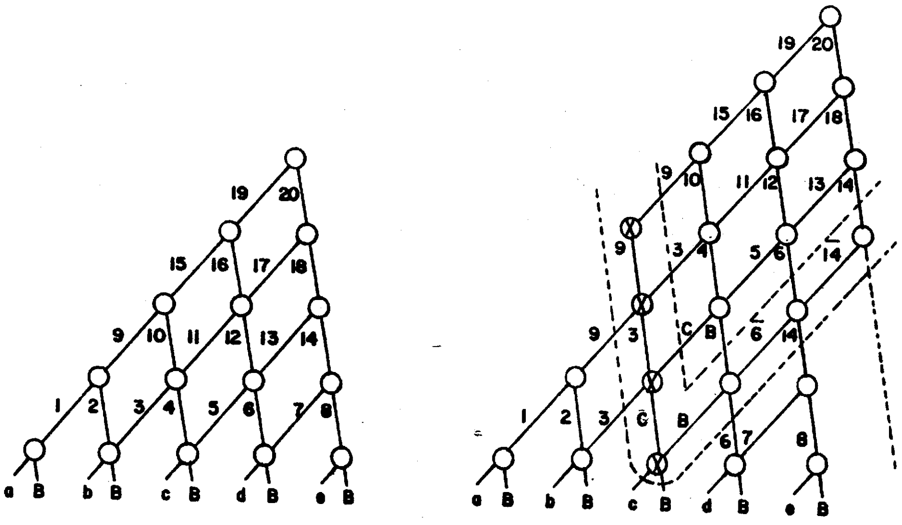


Figure 4.4

Another simple fault-tolerant scheme for *OCA* is explained in the following example.

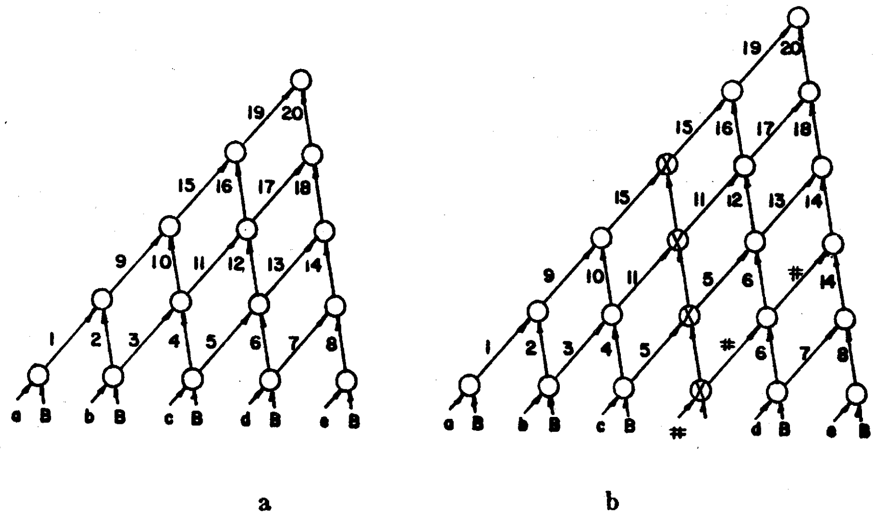


Figure 4.5

In Figure 4.5.b is a faulty *OCA* with the 4th cell failed, which simulates a perfect *OCA* in Figure 4.5.a. The failed cell simply passes the symbols to the right. A special symbol "#" is assumed to be the external input to every failed cell. The input string skips failed cells. If f is the transition function for a perfect *OCA*, the transition function for the corresponding fault-tolerant *OCA* is defined as

$$f'(\alpha,\beta) = \begin{cases} (\beta,\alpha), & \text{if } \alpha \neq \#; \\ f(\alpha,\beta), & \textit{otherwise.} \end{cases}$$

This scheme automatically creates a cut with one unit delay to every edge of the cut. Figure 4.6 is a different time-space diagram of Figure 4.5.b. It emphasizes where the cut is formed by the transition function f'.

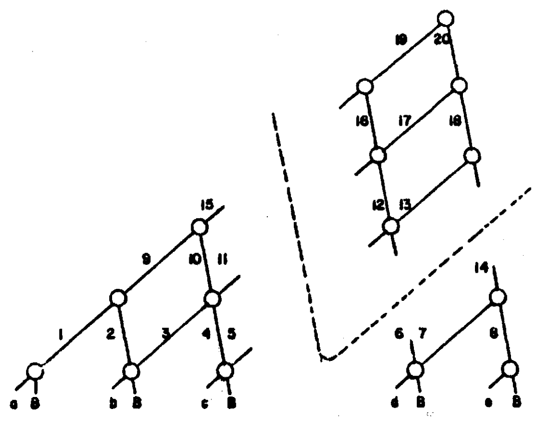


Figure 4.6

For a fault-tolerant bounded *OCA*, the time delay is equal to the number of failed cells involved in the computation. The number of cells used in the computation depends on the distribution of the nonfaulty cells since the number of nonfaulty cells is always equal to the number of input symbols in this scheme.

The unrolling of a one-way cellular ring is equivalent to a trellis [3]. So the fault-tolerant scheme for a *OCA* is very similar to the one for a *OCA*. We omit the details.

5. FAULT-TOLERANT SCHEMES FOR LINEAR CELLULAR AUTOMATA AND ITERATIVE ARRAYS

A cellular automaton (*CA*) ([1], [2], [8], [10], [14]) is shown in Figure 5.1. A *CA* is a linear array of cells. Each cell is a finite automaton. An input to a *CA* is a string of symbols which enter in parallel a sequence of adjacent cells at beginning of the computation. The next state of a cell is the function of the current states of its two nearest neighbors and itself.

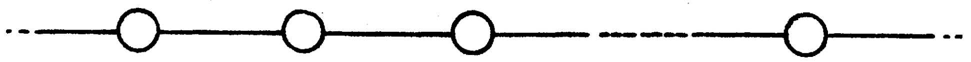


Figure 5.1

Formally, a cellular automaton is a system $A = (C, Q, \Sigma, \delta, F)$, where C is an infinite set of cells which are numbered by IN, Q is a finite set of states,

$\Sigma \subset Q$ is the set of initial states (or, say, inputs), $\delta : Q^3 \to Q$ is the transition function, and $F \subset Q$ is the set of final states . Symbol $\# \in Q$ is the quiescent state.

We use S subscript t, i.e. S_t to denote the cell to state mapping at time t, $S_t : C \to Q$. A cellular automaton (CA) is bounded if δ satisfies $\delta(a,b,c) = \#$ if and only if $b = \#$. In the following, we only consider bounded CA's. Let $c_1, \ldots, c_n$ be the cells which are initially not quiescent. Then $S_{t+1}(c_i) = \delta(S_t(c_{i-1}), S_t(c_i), S_t(c_{i+1}))$, $i \leq i \leq n$, $S_t(c_0) = S_t(c_{n+1}) = \#$.

The fact that an n-time (real time) CA can be simulated by a $2n$-time trellis suggests that fault-tolerant scheme for CA's can be obtained from the one for trellises. However a failed cell in a CA is equivalent to a whole failed column in a trellis. By Theorem 2, the fault-tolerant scheme of previous section will not work in this case. Therefore, we need a different fault-tolerant scheme for the CA's.

We assume that the bypass function for the failed cells of the CA is $p(a,b) = (b,a)$, i.e., the failed cells can simply exchange the state information for the two neighboring cells. The main idea of the fault tolerant scheme for CA is as follows. The state of c_i at time $t+1$ depends on the states of c_{i-1}, c_i, c_{i+1} at time t. If the state of c_{i-1} or c_{i+1} of time t is not available, then we store the partial information at c_i temporarily and will not compute the new state until all of them are ready. The example given in Figure 5.2 shows this idea.

A' (in Figure 5.2.b) looks much more complex than A (in Figure 5.2.a). Actually, each cell of A' only needs two more registers, which we call left register and right register, to remember the status of its two neighbors when waiting for delayed information (because of failed cells) to arrive. The transition function of A' is based on the transition function of A and it follows these principles:

(i) Look at the left (right) register for the state information of the left (right) neighbor.

(ii) If the left (right) register is empty, then get the state information from its left (right) neighbor. If it is not available, enter a "?" to the left (right) register.

(iii) If the left (right) register is a "?", then get the state information from the left (right) neighbor when the left (right) neighbor has a single (not compound) state.

(iv) If all the 3 states are ready then compute the new state according the transition rule, store the new state and empty the left and right registers.

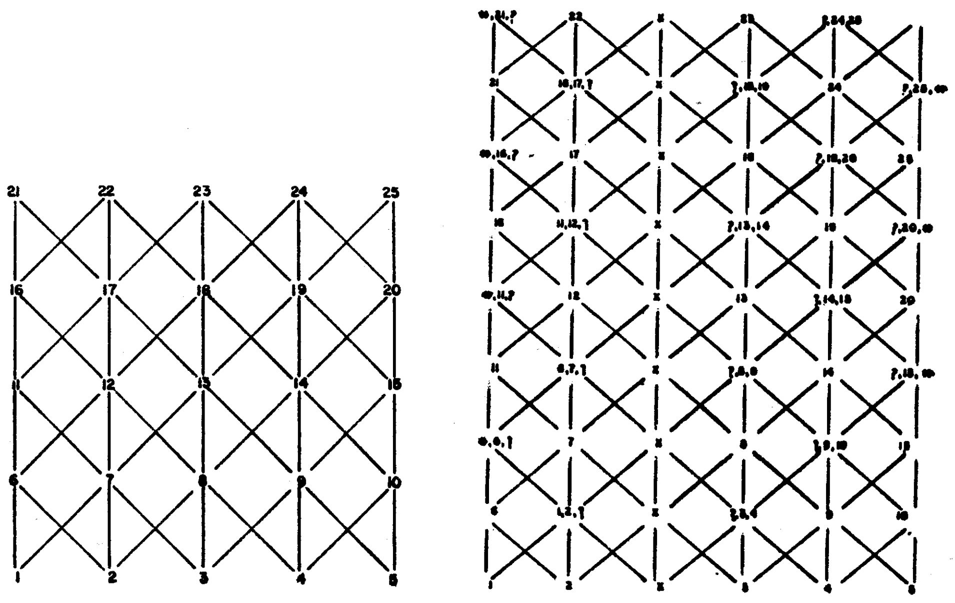


Figure 5.2

Let $A = (C,Q,\Sigma,\delta,F)$ be a cellular automaton without failed cells. We call it a perfect cellular automaton. A fault-tolerant CA $A' = (C',Q',\Sigma,\delta',F)$ based on A is formally constructed as follows:

$$Q' = Q \cup \{(?,q_1,q_2) \mid \{(?,q,?) \mid q \in Q\} \cup \{(q_1,q_2,?) \mid q_1,q_2 \in Q\};$$

δ' : (1) $\delta'(p,q_1,q_2,?),q_3) = \delta(q_1,q_2,q_3)$,

$\delta'(q_1,(?,q_2,q_3),p)$

$\delta'(q_1,(?,q_2,?),q_3) = \delta(q_1,q_2,q_3)$, for $q_1,q_2,q_3 \in Q$, $p \in Q' \cup \{\lambda\}$,

(2) $\delta'(p,q_2,r) = \delta(q_1,q_2,q_3)$, if $p = q_1$ or $(p_1,q_1,?)$

$r = q_3$ or $(?,q_3,r_3)$ for $q_1,q_2,q_3 \in Q$, $p_1,r_3 \in Q'$

(3)

$$\delta'(p,q_2,r) = \begin{cases} (?,q_2,q_3), & \text{if } p \in \{\lambda\} \cup \{?,p_1,p_2 \mid p_1,p_2 \in Q\} \\ & r = q_3 \text{ for } (?,q_3,q), q \in \{?\} \cup Q; \\ (q_1,q_2,?), & \text{if } p \in q_1 \text{ or } (q,q_1,?), q \in \{?\} \cup Q, \\ & r \in \{\lambda\} \cup \{(r_1,r_2,?) \mid r_1,r_2 \in Q\}; \\ (?,q_2,?), & \text{if } p \in \{\lambda\} \cup \{(?,p_1,p_2) \mid p_1,p_2 \in Q\}, \\ & r \in \{\lambda\} \cup \{(r_1,r_2,?) \mid r_1,r_2 \in Q\}. \end{cases}$$

We need to show that, for any finite set $S \subset C'$, A_S' simulates A in order to prove that A' is fault-tolerant and equivalent to A. Assume that $C=\{c_1,c_2,\ldots,c_m,\ldots\}$ and $C'=\{e_1,e_2,\ldots,e_n,\ldots,\}$. Let the notation c^{t_i} or e_i^t to denote the i^{th} cell of A or A' at time t. $T: C' \times N \to N$ is a function to extract the time index, e.g., $T(e_i^t)=t$. We use q_i^t to denote the state of c_i^t, i.e., $q_i^t = S_t(c_i)$.

Let S be an arbitrary finite subset of C' and $C'-S=\{e_{i_1},e_{i_2},\ldots,e_{i_n},\ldots\}$, $i_1<i_2<\cdots<i_n<\cdots$. Function h is used to map the indices of the cells in C to the indices of the cells in $C'-S$, i.e., $h(k)=i_k$. Now we are ready to define a mapping $\rho: C \times P \to C' \times P$ which establishes the basis of the simulation; ρ is defined recursively as follows:

$$\rho(c_i^0)=e_{h(i)}^0, \quad \rho(c_i^{j+1})=e_{h(i)}^k,$$

where

$$k = \max\{T(\rho(c_{i-1}^j))+h(i)-h(i-1),\ T(\rho(c_i^j))+1,\ T(\rho(c_{i+1}^j))+h(i+1)-h(i)\},$$

for $j \geq 0$ and $i \geq 0$.

The construction of A' guarantees that the state of $\rho(c_i^t)$ is the same as the state of c_i^t, for any $i \in P$ and $t \geq 0$. This can be shown by induction on time t. This proof consists of tedious case study, we omit this details but we give the following facts which are used in the induction step of the proof.

Fact 1 Let $A = (C,Q,\Sigma,\delta,F)$ be a CA and $A' = (C',Q',\Sigma,\delta',F)$ be a CA constructed from A by the above scheme. Let $w \in \Sigma^*$ and $|w| = n$. We use q_i^t to denoted the $S_t(c_i)$ of A with w as the input at time 0. Then

(i) if $e_k \in C'$ is a nonfaulty cell, then, for any $m \geq 0$, $S_m(e_k)$ can only have the following forms: q_i^t, $(?, q_i^t, ?)$, $(?, q_i^t, q_{i+1}^t)$ or $(q_{i-1}^t, q_i^t, ?)$, i.e., for some $t \geq m$, all the states stored in the 3 registers of e_k have the same *time-index*;

(ii) if $S_m(e_k) = q_i^t$, then its nearest nonfaulty left neighbor e_j can be in the following states only: q_{i-1}^t, $(q_{i-2}^t, q_{i-1}^e, ?)$, $(?, q_{i-1}^t, ?)$, $(?, q_{i-1}^{t-1}, q_i^{t-1})$;

(iii) if $S_m(e_k) = (?, q_i^t, ?)$ or $(?, q_i^t, q_{i+1}^t)$, then its nearest nonfaulty left neighbor e_j only can be in the following states: q_{i-1}^t, $(?, q_{i-1}^{t-1}, q_i^{t-1})$, $(q_{i-2}^t, q_{i-1}^t, ?)$ or $(?, q_{i-1}^t, ?)$; (the last two are possible only when $j < k-1$)

(iv) if $S_m(e_k) = (q_{i-1}^t, q_i^t, ?)$ then $S_m(e_j)$ can only be in the following possible states: q_{i-1}^{t+1}, $(?, q_{i-1}^t, q_i^t)$, $(q_{i-2}^t, q_{i-1}^t, ?)$; the last case is possible only when $j < k-1$;

(v) for the right nearest nonfaulty neighbor, the possible cases are corresponding to (2), (3), and (4). □

Fact 2 The difference of the time indices of the states (each state can only have one time index, e.g. $(?, q_i^t, q_{i+1}^t)$ has time index t) between two nonfaulty neighbor cells cannot be greater than 1 (or less than -1).

The above results directly lead to the following theorem.

Theorem 5 *For any cellular automaton A, there effectively exist an equivalent fault-tolerant cellular automaton A'.*

The remaining problem on this fault-tolerant CA scheme is how fast is the simulation.

Theorem 6 *Let k be the maximum number of adjacent faulty cells of A' which are involved in the simulation. Then A' can simulate t time units operations of A in $(k+1)t$ time units or less.* □

We now give a informal proof of Theorem 6. Let $e_1, e_2, \ldots, e_m$ be the cells used in the simulation of $c_1, c_2, \ldots, c_n$ of A. Assume $e_{i+1}, \ldots, e_{j-1}$ be the widest band of failed cells and $j - i = k+1$. Then it takes $k+1$ time units to pass the state information between e_i and e_j. This is the maximum communication time between any two nonfaulty neighbors of $e_1, \ldots, e_m$. So, e_i and e_j need $(k+1)t$ time units to simulate t time units operations. The other cells need equal or less time units.

The results on CA's are easily expanded to other linearly connected networks which have parallel inputs.

An iterative array (IA) is a systolic system which is similar to a cellular automaton except the input pattern. Instead of having parallel input, it has a serial one. The input symbols are read by the leftmost cell of the IA and exactly one symbol is read at each time unit. If we relax this requirement and instead we assume that they are read whenever needed, then, with some minor changes, the scheme for the cellular automata also works for the iterative arrays.

To prevent the change of the input pattern, we can first read all the input symbols into the array and then start the computation. This scheme works but it needs more complex transition functions than the simulated iterative array.

Now, we introduce another scheme which is simpler. See Figure 5.3. One step transition of an iterative array can be simulated by two steps of a trellis. A faulty cell in the IA is corresponding to a whole failed diagonal line of the trellis.

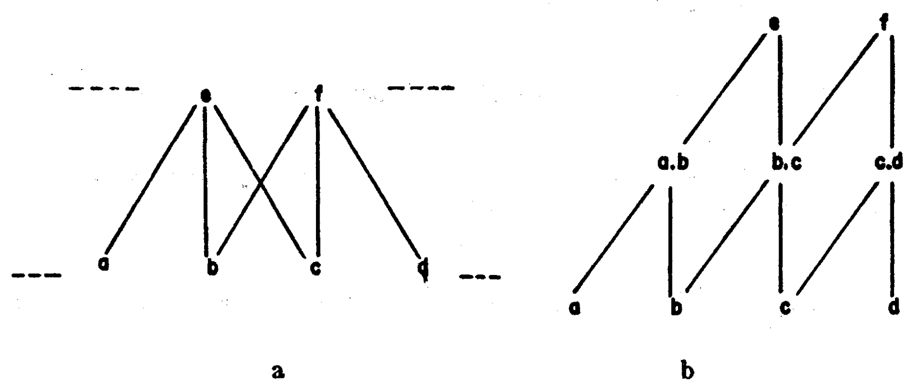
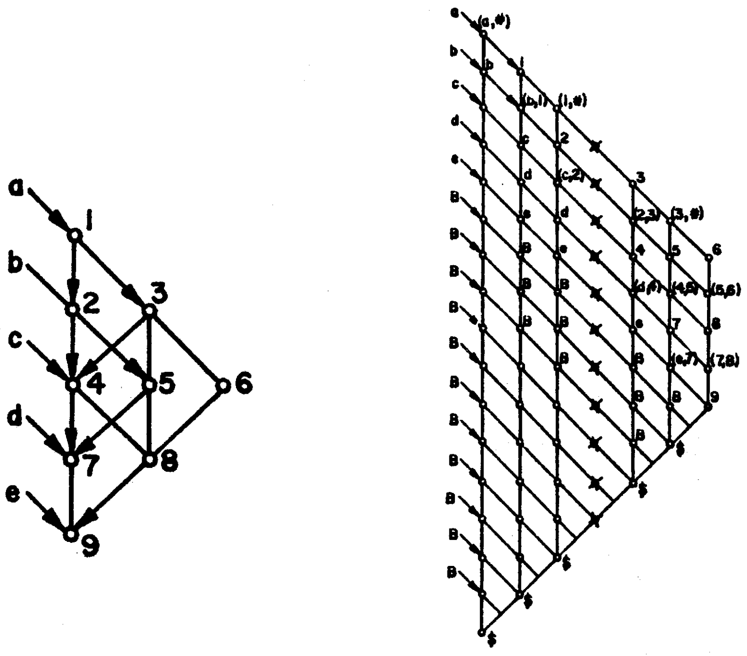


Figure 5.3

We know that the unrolling of a one-way cellular automaton (OCA) is equivalent to a trellis [2]. This scheme consists two phases. In the first phase, it works as an OCA except that the input symbols are read in sequentially by the leftmost cell. This phase is shown at the top 11 steps of Figure 5.4.b. This phase will take $2n+k$ steps to simulate n-step operation (see Figure 5.4) where k is the number of faulty cells involved. In the second phase, the result is passed back to the leftmost cell, which needs $n+k$ steps for the shifting. Totally, we need $3n+2k$ steps to simulate n steps of computation.



a b

Figure 5.4

The bypass function for this scheme is assumed to be $p(a,b) = (b,a)$. The construction of a fault-tolerant IA $A' = (Q',\Sigma,f',F')$ for a given IA $A = (Q,\Sigma,f,F)$ is described as follows. (We assume that $Q \cap \Sigma = \emptyset$, and $B \in \Sigma$ is the blank symbol.)

$$Q' = Q \cup \Sigma \cup \{(a,b) \mid$$

$$F' = \{\$\}$$

$$f'(q_1,q_2,p) = (q_1,q_2), \text{ if } q_1,q_2 \in Q \cup \Sigma, \text{ and } p \in q';$$

$$f'((q_1,q_2)) = f(q_1,q_2,q_3), \text{ if } q_1,q_2,q_3 \in Q, p \in Q';$$

$$f'(B,B,q) = \$, \quad q \in F \cup \{\$\}.$$

The proof of the correctness is omitted. If A accepts a string w in n time units, then A' accepts w in $3n+2k$ time units, where k is the number of failed cells involved. So, we have the following theorem.

Theorem 7 *For any iterative array A, there exists an equivalent fault-tolerant iterative array A'.*

This result can also be expanded to all linearly connected systolic networks with serial inputs.

6. FAULT TOLERANT UNIDIRECTIONAL RING SIMULATING BIDIRECTIONAL RING OR BIDIRECTIONAL ARRAY

Our notion of fault-tolerant scheme used in previous sections requires an algorithm that converts a system of given type into an equivalent fault-tolerant system of the *same* type. However, it is reasonable to drop the requirement that the simulating system must be of the same type. This is especially true if the simulating system is even of a simpler type.

In [3] it was shown how topological transformations of unrollings (time-space diagrams) of systolic systems can be used to demonstrate the equivalence of systolic systems. As an example it was shown that a bidirectional ring or bidirectional array can be simulated by a twice slower unidirectional ring of the same size. We demonstrate this result directly (without using the equivalence of trellises and one-dimensional arrays). Then we use the fact that there exists a fault tolerant scheme for unidirectional rings (the simplest known fault-tolerance scheme) to obtain the corollary that every bidirectional ring or array (linear CA) can be converted into an equivalent fault-tolerant undirectional ring.

Theorem 8 *Given a bidirectional ring (linear CA) we can construct a unidirectional ring of the same size that simulates every m-step computation on A in $2m$ steps.*

Proof: We show the transformation for the linear CA, for bidirectional ring the situation is the same except that all the edges including the dotted ones in Figure 6.8 are used.

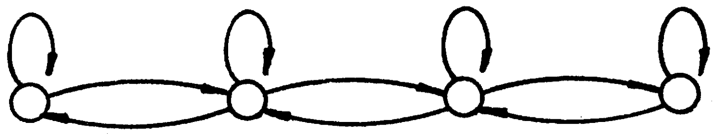


Figure 6.1

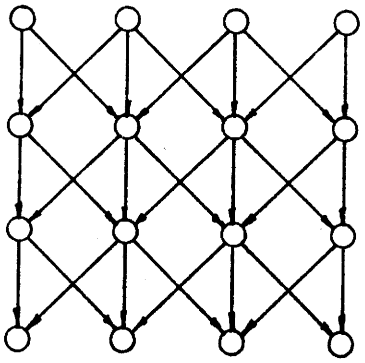


Figure 6.2

A linear *CA* is shown in Figure 6.1 and its unrolling (time-space diagram of a 3-step computation) in Figure 6.2. The self-loops in Figure 6.1 indicate that each processor has a memory. The digraphs in Figures 6.2 and 6.4 are clearly isomorphic, thus (see [3]) the networks in Figures 6.1 and 6.3 are equivalent. If we identify, in top-down order, the odd rows of Figure 6.6 with all the rows of Figure 6.4 we see that for every edge in Figure 6.4 we have a path of length two connecting the corresponding nodes of the digraph in Figure 6.6. Thus by Lemma 2 in [3], the network in Figure 6.5 can (1,2)-simulate the network in Figure 6.3.

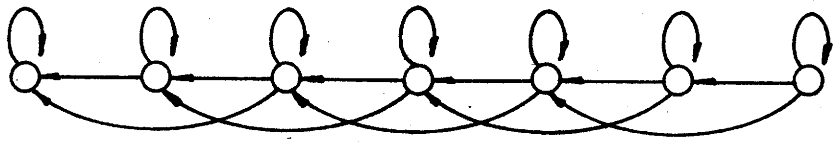


Figure 6.3

Finally, we observe that the digraphs in Figures 6.6 and 6.8 (without the dotted edges) are isomorphic. Since the digraph in Figure 6.8 (with the dotted edges) is the unrolling of a unidirectional ring shown in Figure 6.7 we conclude that every

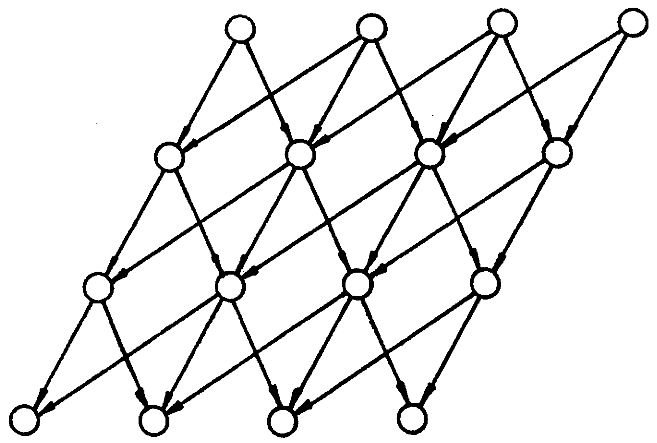


Figure 6.4

linear array can be (1,2)-simulated on a unidirectional ring. When modifying the construction for the simulation of a bidirectional ring we will make use of all the edges, including the dotted ones, in Figure 6.8. □

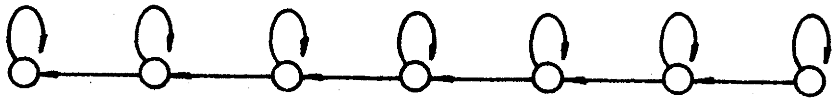


Figure 6.5

As discussed in Section 4 and also in [11] given a unidirectional ring A with n cells it is easy to construct a fault-tolerant unidirectional ring B with $n+k$ cells which simulates any $T(n)$-step computation on A in at most $cT(n)$ steps providing that at most k of its cells are faulty and that no more than c consecutive cells are faulty. Thus we have the following:

Corollary *Given a bidirectional ring A (linear CA) with n processors we can construct a unidirectional (cellular) ring B with $n+k$ processor that simulates every $T(n)$ computation on A in $O(T(n+k))$ steps and is fault-tolerant with respect to Ω, where Ω is the collection of all subsets of cells of A of cardinality at most k.*

Note If the processors (cells) of B are made more complicated so that each cell can simulated more than one cell of A (essentially overlapping two or more

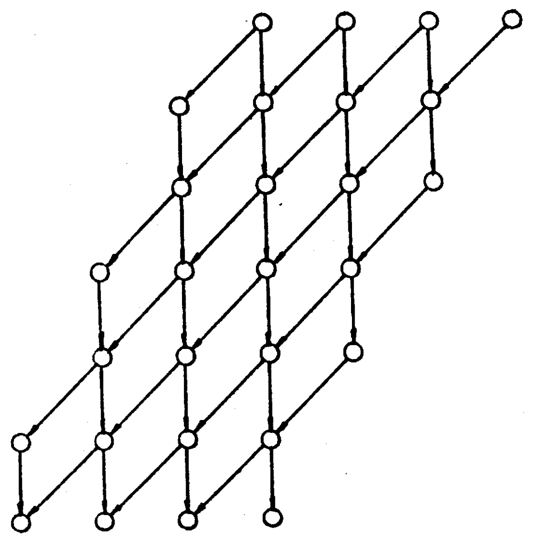


Figure 6.6

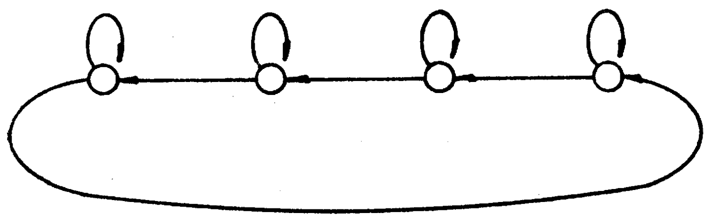


Figure 6.7

"tracks" of cells) we can make B fault tolerant with respect to a larger set of subsets Ω or alternatively we can choose $k \leq 0$, i.e. the ring B can be of the same (or even smaller) size as the simulated linear array (CA) A.

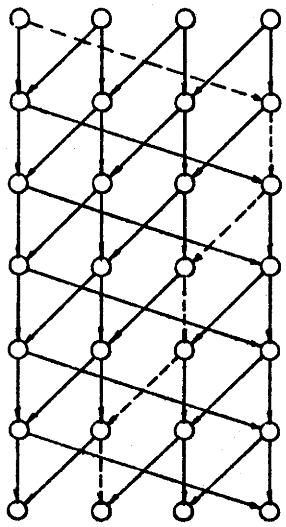


Figure 6.8

REFERENCES

[1] W. Bucher and K. Culik II, On Real Time and Linear Time Cellular Automata, *RAIRO, Inf. Theoretique,* to appear.

[2] C. Choffrut and K. Culik II, On Real Time Cellular Automata and Trellis Automata, *Acta Informatica,* to appear.

[3] K. Culik II and I. Fris, Topological Transformations as a Tool in the Design of Systolic Networks, Technical Report, CS-84-11, April 1984, Univ. of Waterloo.

[4] K. Culik II, J. Gruska and A. Salomaa, Systolic Trellis Automata, Part I, *Intern. J. Computer Math. 15,* (1984), pp.195-212.

[5] K. Culik II, J. Gruska and A. Salomaa, Systolic Trellis Automata, Part II, *Intern. J. Computer Math. 16,* (1984), pp. 3-22.

[6] C.R. Dyer, One Way Bounded Cellular Automata, *Inform. and Control 44,* (1980), pp. 261-281.

[7] M. Harao and S. Noguchi, Fault Tolerant Cellular Automata, *Journal of Computer and System Sciences11,* (1975), pp.171-185.

[8] F.C. Hennie, Iterative Arrays of Logical Circuits, *MIT Press,* Cambridge, Mass., 1961.

[9] O.H. Ibarra, S.M. Kim and S. Moran, Sequential Machine Characterizations of Trellis and Cellular Automata and Applications, unpublished manuscript, 1983.

[10] S.P. Kosarajn, On Some Open Problems in the Theory of Cellular

Automata, *IEEE Trans. Computers C-30*, (1974), pp.561-565.

[11] H.T. Kung and M.S. Lam, Fault-Tolerant and Two-Level Pipelining in VLSI Systolic Arrays, *MIT Conf. on Advanced Research in VLSI*, Jan. 1984.

[12] H.T. Kung, Why Systolic Architecture?, *Computer 15*, 1 (1982), pp.37-46.

[13] H. Nishio and Y. Kobuchi, Fault Tolerant Cellular Spaces, *Journal of Computer and System Sciences 11* (1975), pp. 150-170.

[14] A.R. Smith III, Real-Time Languages Recognition by One-Dimensional Cellular Automata, *J. of Computer and System Science 6*, (1972), pp. 233-253.