KYBERNETIKA — VOLUME 35 (1999), NUMBER 1, PAGES 39 - 55

STRUCTURED REDUNDANCY FOR FAULT TOLERANCE IN STATE–SPACE MODELS AND PETRI NETS[1]

CHRISTOFOROS N. HADJICOSTIS AND GEORGE C. VERGHESE

The design and implementation of systems in state form has traditionally focused on *minimal* representations which require the least number of state variables. However, "structured redundancy" – redundancy that has been intentionally introduced in some systematic way – can be extremely important when fault tolerance is desired. The redundancy can be used to detect and correct errors or to guarantee desirable performance despite hardware or computational failures. Modular redundancy, the traditional approach to fault tolerance, is prohibitively expensive because of the overhead in replicating the hardware. This paper discusses alternative methods for systematically introducing redundancy in state-space systems. Our approach consists of mapping the state space of the original system into a redundant space of higher dimension while preserving the properties of the original system in some encoded form within this larger space. We illustrate our approach by focusing primarily on linear time-invariant (LTI) systems in state form. We provide a complete characterization of the class of appropriate redundant systems and demonstrate through several examples ways in which our framework can be used for achieving fault tolerance. We also discuss appropriate error models and outline the extension of our approach to Petri nets.

1. INTRODUCTION

In this paper we explore a design methodology for fault-tolerant systems in state form. Our approach is based on mapping the state of the original system into a larger, redundant space, while at the same time preserving the properties and information contained in the original system – perhaps in some encoded form. The redundancy we add into the system can be used to achieve error correction or robust performance despite hardware failures. Even though we focus on linear time-invariant (LTI) state-space systems and Petri nets, our approach is general and can be used in a variety of settings.

[1]This work has been supported in part by the Department of the Navy, Office of the Chief of Naval Research, contract number N00014-93-1-0686 as part of the Advanced Research Projects Agency's RASSP program, and also by fellowships from the National Semiconductor Corporation and the Grass Instrument Company.

Traditional system design has aimed at the realization of *minimal* systems, i.e., systems that require minimal resources (these resources could be hardware, computation time, power consumption, system dimension, etc.). There has, however, also been a long-standing interest in redundant systems that are *fault-tolerant*. The traditional, but rather inefficient, way of designing fault-tolerant systems is to use N-modular hardware redundancy, [35]: by replicating the original system N times, we perform the desired function multiple times in parallel. The outputs of all replicas are compared, and the final result is chosen based on what the majority of them has agreed upon. Also, those in the minority are declared faulty.

Research in communications has extensively explored alternative, more efficient ways of utilizing redundancy for error detection and correction. Examples of such efficient schemes are the error correcting codes that are used when one transmits digital data through an imperfect channel, [36]. In more complex systems that involve not only simple transmission of the data but also some form of processing on the data (e.g., computational or signal processing systems), the application of such error correcting ideas is more challenging. Work in this direction includes *arithmetic* codes (see, for example, [29]) and algorithm-based fault tolerance (ABFT) techniques (introduced by Abraham, [18, 21, 26], and subsequently developed by others). These techniques have been quite successful, but each time they have to be cleverly tailored to the specific application under consideration.

More broadly applicable and systematic approaches for introducing redundancy in computational systems were studied recently by Beckmann, [3, 4], and later by us, [15, 16]. Beckmann's work focused on computations that can be modeled as abelian group operations, and used group homomorphisms both to introduce redundancy and to analyze its properties. Our work extended Beckmann's framework and analyzed operations that can be modeled as occurring in *semigroups* or *semirings*. Even though this is a very broad setting, we have been able to generalize most of Beckmann's results and to develop an algebraic framework for analyzing a large class of fault-tolerant computational systems.

This paper describes a mathematical framework in the spirit of [3, 16] for the design of fault-tolerant LTI state-space systems and Petri nets. Our approach consists of mapping the original state vector into a higher dimensional space in a way that preserves the evolution and properties of the original system. This results in an embedding of the original system into a larger, redundant system. In the case of LTI state-space systems we are able to completely characterize all possible redundant systems and to illustrate that our method essentially amounts to augmenting the original system with redundant *modes* that are *unreachable* but *observable* under fault-free conditions. Because these additional modes are not excited initially, they manifest themselves only when a fault takes place. Our characterization turns out to be a special case of results on LTI system "inclusion" treated in [19], although the issue of creating redundancy for fault tolerance does not seem to have been a motivation for [19]. We describe these results and present examples related to fault tolerance in Section 2. In Section 3 we apply this approach to Petri nets.

2. REDUNDANT LTI SYSTEMS

Linear time-invariant systems in state form constitute a well studied class of dynamic systems with a variety of applications, such as filter design, system simulation and model-based control, [22, 23, 31]. Although our discussion is focused on the discrete-time case, most of our results and examples can be translated to the continuous-time case in a straightforward manner.

An LTI system is represented in state form by the following pair of equations:

$$x[k+1] = Ax[k] + Bu[k] , \quad (1)$$

$$y[k] = Cx[k] + Du[k] , \quad (2)$$

where k is the discrete-time index, $x[k]$ is the n-dimensional *state vector*, $u[k]$ is the m-dimensional *input vector*, and $y[k]$ is the p-dimensional *output vector*. Eq. (1) is referred to as the *state evolution* equation and eq. (2) is the *output* equation; A, B, C, and D are constant matrices of appropriate dimensions. We assume that the entries of all vectors and matrices are real numbers.

One can obtain equivalent state-space models (with n-dimensional state vector $x'[k]$) through similarity transformation, [22, 23]:

$$\begin{aligned} x'[k+1] &= (T^{-1}AT)x'[k] + (T^{-1}B)u[k] \equiv A'x'[k] + B'u[k] , \\ y[k] &= (CT)x'[k] + Du[k] \equiv C'x'[k] + D'u[k] , \end{aligned}$$

where T is an invertible $n \times n$ matrix such that $x[k] = Tx'[k]$. The initial conditions for the transformed system can be obtained as $x'[0] = T^{-1}x[0]$. Systems related in such a way are known as *similar* systems.

Given an input-output specification of an LTI system, there exist many possible ways of *realizing* the specification, that is, relating it to a particular state-space representation as in eqs. (1) and (2) above. A realization that uses the minimum possible number of state variables is called *minimal*. We are interested in systematically adding redundancy to such minimal realizations in order to achieve error detection and correction.

2.1. Systematic introduction of redundancy

In order to provide error detection and correction to an LTI state-space system S of dimension n, we implement a redundant state-space system $\mathcal{S}$ of dimension η ($\eta \equiv n + d$, $d > 0$). Given the original input, $\mathcal{S}$ evolves so that its state vector at each time step provides complete information about the corresponding state of the original system S. The added redundancy can be used for error protection. We develop this claim in more detail next. For the rest of this paper, we essentially ignore the output equation (2) and focus on the state evolution equation (1).

Let the desired state evolution equation of the original system S be given by eq. (1). We wish to implement a redundant system $\mathcal{S}$ with state evolution

$$\xi[k+1] = \mathcal{A}\xi[k] + \mathcal{B}u[k] \quad (3)$$

ensuring that, at every time step k, the state vector $x[k]$ of S can be recovered from $\xi[k]$ through a *constant* $n \times \eta$ *decoding* matrix L, i.e.,

$$x[k] = L\xi[k] \quad \text{for all } k \, .$$

(Note that, under the assumptions so far, the redundant system $\mathcal{S}$ can be regarded as a *cover* for S. In the language of finite automata, an automaton $\mathcal{S}$ is a cover for an automaton S if, given the same input, there exists a mapping of the state of $\mathcal{S}$ at any given time to the corresponding state of S, [13].)

In order to achieve fault-tolerance within this LTI setting, we impose a design requirement on the states of the redundant system $\mathcal{S}$: there should exist a constant linear mapping from each state in S to a state in $\mathcal{S}$. This is a natural constraint for using the redundancy in $\mathcal{S}$ in some useful way. This linear mapping can be represented by an $\eta \times n$ *encoding* matrix Φ so that

$$\xi[k] = \Phi x[k] \quad \text{for all } k \, .$$

Under the above assumptions, $L\Phi = I_n$ (where I_n is the $n \times n$ identity matrix). Note that this equation by itself does not uniquely fix L or Φ. Fault detection is straightforward: since the redundant state vector must be in the column space of Φ under fault-free conditions, all we need to check is that at each time step k, $\xi[k]$ lies in the column space of Φ. Equivalently, we can check that $\xi[k]$ is in the null space of an appropriate *parity check matrix* Θ, so $\Theta\xi[k] = 0$ under fault-free conditions. We illustrate ways of obtaining the matrix Θ later in this section.

Theorem 1. In the setting described above, a system $\mathcal{S}$ (of dimension $\eta \equiv n + d$, $d > 0$) is a redundant version of S if and only if it is *similar* to a *standard* redundant system $\mathcal{S}_\sigma$ whose state evolution equation is given by

$$\xi_\sigma[k+1] = \begin{bmatrix} A & A_{12} \\ 0 & A_{22} \end{bmatrix} \xi_\sigma[k] + \begin{bmatrix} B \\ 0 \end{bmatrix} u[k] \, . \tag{4}$$

Here, A and B are the matrices in eq. (1), A_{22} is a $d \times d$ matrix that describes the added modes, and A_{12} is an $n \times d$ matrix that describes the coupling between the redundant and non-redundant modes. Associated with this standard redundant system are the standard decoding, encoding, and parity check matrices:

$$L_\sigma = \begin{bmatrix} I_n & 0 \end{bmatrix} \, , \quad \Phi_\sigma = \begin{bmatrix} I_n \\ 0 \end{bmatrix} \, , \quad \Theta_\sigma = \begin{bmatrix} 0 & I_d \end{bmatrix} \, .$$

Proof. Let $\mathcal{S}$ be a redundant version of S. From $L\Phi = I_n$, we know that L is full-row-rank and Φ is full-column-rank. Furthermore, there exists an invertible $\eta \times \eta$ matrix $\mathcal{T}$ such that $L\mathcal{T} = \begin{bmatrix} I_n & 0 \end{bmatrix}$ and $\mathcal{T}^{-1}\Phi = \begin{bmatrix} I_n \\ 0 \end{bmatrix}$. If we apply the transformation $\xi[k] = \mathcal{T}\xi'[k]$ to system $\mathcal{S}$, we obtain a similar system $\mathcal{S}'$ with

decoding mapping $L' = LT = \begin{bmatrix} I_n & 0 \end{bmatrix}$ and encoding mapping $\Phi' = T^{-1}\Phi = \begin{bmatrix} I_n \\ 0 \end{bmatrix}$. The state evolution of the redundant system $\mathcal{S}'$ is given by

$$\xi'[k+1] = (T^{-1}\mathcal{A}T)\xi'[k] + (T^{-1}\mathcal{B})u[k] \equiv \mathcal{A}'\xi'[k] + \mathcal{B}'u[k] \,. \tag{5}$$

For all time steps k, and under fault-free conditions, $\xi'[k] = \Phi' x[k] = \begin{bmatrix} x[k] \\ 0 \end{bmatrix}$. Combining the state evolution equations of the original and redundant systems (eqs. (1) and (5) respectively), we see that

$$\begin{bmatrix} Ax[k] + Bu[k] \\ 0 \end{bmatrix} = \begin{bmatrix} \mathcal{A}'_{11} & \mathcal{A}'_{12} \\ \mathcal{A}'_{21} & \mathcal{A}'_{22} \end{bmatrix} \begin{bmatrix} x[k] \\ 0 \end{bmatrix} + \begin{bmatrix} \mathcal{B}'_1 \\ \mathcal{B}'_2 \end{bmatrix} u[k] \,.$$

By setting the input $u[k] \equiv 0$ for all k, we conclude that $\mathcal{A}'_{11} = A$ and $\mathcal{A}'_{21} = 0$. With the input now allowed to be non-zero, we deduce that $\mathcal{B}'_1 = B$ and $\mathcal{B}'_2 = 0$. The system $\mathcal{S}'$ is therefore in the form of the standard system $\mathcal{S}_\sigma$ in eq. (4) with appropriate decoding and encoding matrices. The check matrix can be $\Theta' = \begin{bmatrix} 0 & P \end{bmatrix}$, where P is *any* invertible $d \times d$ matrix; a trivial similarity transformation will ensure that the parity check matrix takes the form $\begin{bmatrix} 0 & I_d \end{bmatrix}$, while keeping the system in the standard form $\mathcal{S}_\sigma$ in eq. (4) – except with $A_{12} = \mathcal{A}'_{12}P$ and $A_{22} = P^{-1}\mathcal{A}'_{22}P$. The decoding, encoding and check matrices are then as claimed in the statement of the theorem.

The converse, namely that if $\mathcal{S}$ is similar to a standard $\mathcal{S}_\sigma$ as in (4) then it is a redundant version of (1), is easy to show. □

The above theorem establishes a complete characterization of all possible fault-tolerant designs (subject to our restrictions) of a given LTI state-space model. The additional modes introduced by the redundancy never get excited under fault-free conditions because they are initialized to 0 and they are unreachable from the input. Due to the existence of the coupling matrix A_{12}, the additional modes are not necessarily unobservable through the decoding matrix. The above theorem (but stated for the continuous-time case) essentially appears in [19], although the proof is different and the motivation apparently very different.

2.2. Error model for LTI systems

A detailed discussion of error detection and correction requires a specific error model. In this section we describe the sorts of hardware failures that might take place in the implementation of our systems, and the way we reflect these *faults* into our theoretical framework (i.e., we describe our error model). We study two kinds of hardware faults: *transient* (soft) and *permanent* (hard) faults, [3]. A transient fault at time step k occurs only at that particular time step, but disappears at the following ones. Therefore, if the errors are corrected before the initiation of step $k+1$, the system will resume its normal mode of operation. A permanent fault, on the other hand, causes errors at all remaining time steps. Clearly, a permanent fault can be treated as a transient fault for all remaining time steps (assuming error

correction at every time step), but in certain cases one can deal with it in more efficient ways (e.g., reconfiguration).

We assume that we implement our LTI systems using delays (memory elements), adders and gains (multipliers) that we interconnect in appropriate ways. These realizations can be represented using delay-adder-gain diagrams, or signal flow graphs. The same state-space description (matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ for the redundant system) corresponds to a number of different delay-adder-gain diagrams; consequently, it can have a number of different hardware realizations, [31]. This makes the connection with hardware failures more complicated because in certain implementations a single fault in a multiplier or an adder can corrupt more than a single entry in the matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ (and more than one state variable). If we assume that we implement our systems using delay-adder-gain diagrams in which the longest delay-free path is of length one, then the multiplier gains are directly reflected as the entries in the matrices of the state-space description, [31]. We can then model faults in the multipliers (and in the adders) as corruptions in individual entries of the matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$. This is the assumption that we make when we analyze the examples in the next section[2].

The importance of the actual hardware implementation can also be seen from the following example: if our redundant system is directly implemented in the standard form (4), with the parity check matrix $\Theta_\sigma = \begin{bmatrix} 0 & I_d \end{bmatrix}$, then the redundancy is quite useless; under the assumptions of the previous paragraph, the only faults that are detected are ones that directly affect the redundant modes of the system at time step k (because these additional modes are not influenced by the original modes or the input). This is pointless, because our objective is to use the redundancy to protect the original system, not to protect the redundancy itself. However, systems that are *similar* to the standard one can be designed to provide efficient error protection.

2.3. Examples of fault-tolerant LTI systems

Triple Modular Redundancy: Triple modular redundancy (TMR) maintains three separate copies of the original system. These copies (modules) use *separate hardware* and operate identically under fault-free conditions. By comparing their state vectors at a given time step, one is able to detect errors. In fact, errors in one of the state vectors can easily be corrected using a nonlinear, but otherwise simple, voting scheme: we select the state vector agreed upon by two or more systems. TMR in the LTI state-space case corresponds to a system of the form

$$\xi[k+1] \equiv \begin{bmatrix} x^1[k+1] \\ x^2[k+1] \\ x^3[k+1] \end{bmatrix} = \begin{bmatrix} A & 0 & 0 \\ 0 & A & 0 \\ 0 & 0 & A \end{bmatrix} \xi[k] + \begin{bmatrix} B \\ B \\ B \end{bmatrix} u[k] , \qquad (6)$$

where the initial conditions are chosen so that the state vectors $x^1[k]$, $x^2[k]$ and $x^3[k]$ of the three subsystems evolve in the same way as in the original one (i.e., $x^1[0] =$

[2] A future step is to study more general descriptions, such as *factored state variable* descriptions, [31]. It is also possible to accommodate for implementations that are based on more general delay-adder-gain diagrams by looking at the technique in our adaptive decoding example in the next section, or by employing the computation trees in [10].

$x^2[0] = x^3[0] = x[0]$). The encoding matrix Φ is given by $\begin{bmatrix} I_n & I_n & I_n \end{bmatrix}^T$, whereas the decoding mapping L can be $\begin{bmatrix} I_n & 0 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & I_n & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 0 & I_n \end{bmatrix}$, or others (e. g., convex combinations). In this example we assume that $L = \begin{bmatrix} I_n & 0 & 0 \end{bmatrix}$. The parity check matrix Θ can be $\begin{bmatrix} -I_n & I_n & 0 \\ -I_n & 0 & I_n \end{bmatrix}$. When a non-zero entry appears on the upper (respectively, lower) half of the $2n$-dimensional vector $\Theta\xi[k]$, we know that a fault took place in subsystem 2 (respectively, 3). When non-zero entries appear in both the top and bottom half-vectors, then a fault exists in subsystem 1.

The TMR system is easily shown to be similar to

$$\xi_\sigma[k+1] = \begin{bmatrix} A & 0 & 0 \\ 0 & A & 0 \\ 0 & 0 & A \end{bmatrix} \xi_\sigma[k] + \begin{bmatrix} B \\ 0 \\ 0 \end{bmatrix} u[k] ,$$

which is of the form described in the theorem of the previous section. The initial conditions are now $\xi_\sigma[0] = \begin{bmatrix} x[0] & 0 & 0 \end{bmatrix}^T$ where $x[0]$ is the initial condition associated with the original system. Note that all modes of the original system are replicated twice ($A_{22} = \begin{bmatrix} A & 0 \\ 0 & A \end{bmatrix}$) and there is no coupling ($A_{12} = 0$). The check matrix for the standard system is $\Theta_\sigma = \begin{bmatrix} 0 & I_n & 0 \\ 0 & 0 & I_n \end{bmatrix}$ as expected.

Once the encoding matrix Φ is fixed, the additional freedom in choosing the decoding matrix L can be used to our advantage. For example, when our checking procedure detects *permanent* faults in the first subsystem, we can change our decoding matrix from $L = \begin{bmatrix} I_n & 0 & 0 \end{bmatrix}$ to $L = \begin{bmatrix} 0 & I_n & 0 \end{bmatrix}$. This will ensure that the overall output is still correct. In fact, this idea is generalized quite a bit in our adaptive decoding analysis later in this section.

Linear Coding: Using our framework, we can develop schemes that provide detection and correction of transient faults. The following is a simple motivating example to illustrate the idea. Let the original system be

$$x[k+1] = \begin{bmatrix} .2 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 \\ 0 & 0 & .1 & 0 \\ 0 & 0 & 0 & .6 \end{bmatrix} x[k] + \begin{bmatrix} 3 \\ -1 \\ 7 \\ 0 \end{bmatrix} u[k] .$$

To protect this system against single transient errors in the state variables or the matrix entries, we decide to use three additional modes; more specifically, the standard redundant system looks like

$$\xi_\sigma[k+1] = \begin{bmatrix} .2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .3 \end{bmatrix} \xi_\sigma[k] + \begin{bmatrix} 3 \\ -1 \\ 7 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u[k] .$$

For error detection, we need to check whether $\Theta_\sigma \xi_\sigma[k]$ is 0 (where $\Theta_\sigma = \begin{bmatrix} 0 & I_3 \end{bmatrix}$). However, as we argued earlier, redundant systems in standard form cannot be used for detecting or correcting errors in the *original* modes: given a faulty state vector $\xi_\sigma^f[k]$, the fact that $\Theta_\sigma \xi_\sigma^f[k] \neq 0$ will simply mean that an error took place in the calculation of the *redundant* modes. What we would really like is to protect against errors that appear in the original modes. One way to achieve this is to employ a system similar to the standard redundant system, but with the following parity check matrix:

$$\Theta = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{7}$$

This choice of Θ is motivated by the structure of Hamming codes in communications, see [36]. With a suitable similarity transformation $\mathcal{T}$ (so that $\Theta\mathcal{T} = \Theta_\sigma$), the corresponding redundant system is

$$\xi[k+1] = \begin{bmatrix} .2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .6 & 0 & 0 & 0 \\ 0 & -.3 & .1 & 0 & .2 & 0 & 0 \\ .3 & 0 & 0 & -.1 & 0 & .5 & 0 \\ .1 & 0 & .2 & -.3 & 0 & 0 & .3 \end{bmatrix} \xi[k] + \begin{bmatrix} 3 \\ -1 \\ 7 \\ 0 \\ -9 \\ -2 \\ -10 \end{bmatrix} u[k]. \tag{8}$$

This system can detect and locate transient faults that cause the value of a single state variable to be incorrect at a particular time step. To do this, we check for non-zero entries in the vector $\theta[k] \equiv \Theta\xi[k]$. If, for example, $\theta_1[k] \neq 0$, $\theta_2[k] \neq 0$, and $\theta_3[k] \neq 0$, then the value of $\xi_1[k]$ is corrupted; if $\theta_1[k] \neq 0$, $\theta_2[k] \neq 0$, and $\theta_3[k] = 0$, then a fault has corrupted $\xi_2[k]$; and so forth. Once the erroneous variable is located, we can correct it using any of the parity equations in which it appears. For example, if $\xi_2[k]$ is corrupted, we can calculate the correct value by setting $\xi_2[k] = -\xi_1[k] - \xi_3[k] - \xi_5[k]$ (i.e., using the first parity equation). If the faults are transient, the operation of the system will resume normally in the following steps.

A simple checksum approach appeared in [18] under the name "state variable filter". A *real coding* scheme with the ability to detect and correct single errors was developed in [10]. Both schemes are special cases of our framework. In [10] (as well as in [9] where one of the authors of [10] analyzes the continuous-time case), they do not consider different similarity transformations and they do not permit the additional modes to be non-zero. Clearly, our framework is more general: for example, by taking advantage of additional non-zero modes one can devise stable fault-tolerant schemes for continuous-time systems[3] or construct schemes in which checking can be done non-concurrently (e.g., periodically).

[3] In [9], they used "negative feedback" or "lossy integrators" to deal with the stability problem. Our use of non-zero redundant modes avoids this issue completely.

Adaptive Decoding: In the TMR example of eq. (6), a permanent fault in any subsystem can be detected using the check matrix $\Theta = \begin{bmatrix} -I_n & I_n & 0 \\ -I_n & 0 & I_n \end{bmatrix}$. The corrupted state variable(s) can be corrected by simple majority voting or by using any of the relevant parity equations. However, when the fault is permanent, we would like to be able to avoid the overhead of error correction at each time step. In the TMR case, this can be done in a straightforward way: for example, once a fault permanently corrupts the first subsystem (by corrupting entries in its A or B matrices), we can switch our decoding matrix from $L = [\ I_n \quad 0 \quad 0\ ]$ to $L = [\ 0 \quad I_n \quad 0\ ]$ (or $L = [\ 0 \quad 0 \quad I_n\ ]$ or others) and ignore the parity checks that involve variables in the first subsystem. This ensures that the output of the redundant system is still correct. We can continue to perform error detection, but have lost the ability to do error correction. We now formalize and generalize this idea.

Consider again the redundant system $\mathcal{S}$ whose state evolution equation is given by eq. (3). Under fault-free conditions, $x[k] = L\xi[k]$ and $\xi[k] = \Phi x[k]$ for all k. Suppose that we implement this system using a delay-adder-gain interconnection with delay-free paths of unit length. A permanent fault in a multiplier of the system manifests itself as a corrupted entry in matrices $\mathcal{A}$ or $\mathcal{B}$: the ith state variable $\xi_i[k]$ (and other $\xi_j[\cdot]$ at later steps) will be corrupted if some of the entries[4] $\mathcal{A}(i, l_1)$ and/or $\mathcal{B}(i, l_2)$ (l_1 in $\{1, 2, ..., n\}$, l_2 in $\{1, 2, ..., m\}$) are corrupted right after time step $k-1$. We assume that we can locate the faulty state variable through the use of some linear error correcting scheme as in the previous example. We do not have control over the entries in $\mathcal{A}$ and $\mathcal{B}$, but we are allowed to adjust the decoding matrix L to a new matrix L_a. We would like to know which entry corruptions can be tolerated, and how to choose L_a.

The first step is to find out which state variables will be corrupted eventually. If at time step k_0 we detect a corruption at the ith state variable, then we know that at time step $k_0 + 1$, state variable $\xi_i[k_0]$ will corrupt the state variables that depend on it (let M_{i_1} be the set of indices of these state variables – including i); at time step $k_0 + 2$, the state variables with indices in set M_{i_1} will corrupt the state variables that depend on them; let their indices be in set M_{i_2} (which includes M_{i_1}); and so on. Eventually, the final set of indices for all corrupted state variables is given by the set M_{i_f} (note that $M_{i_f} = M_{i_\eta} = M_{i_1} \cup M_{i_2} \cup M_{i_3} ... \cup M_{i_\eta}$). The sets of indices M_{i_f} for all i in $\{1, 2, ..., \eta\}$ can be *pre-calculated* in an efficient manner by computing $R(\mathcal{A})$, the *reachability matrix* of $\mathcal{A}$, as outlined in [27].

Once we have detected a fault at the ith state variable, our new decoding matrix L_a (if it exists) should not make use of state variables with indices in M_{i_f}. Equivalently, we ask the question: does there exist a decoding matrix L_a such that $L_a\Phi_a = I_n$? Here, Φ_a is the same as the original encoding matrix Φ except that $\Phi_a(j, l)$ is set to zero for all l in $\{1, 2, ..., n\}$ and j in M_{i_f}. If Φ_a is full-column-rank, such an L_a exists (any L_a that satisfies $L_a\Phi_a = I_n$ is suitable). In this case our redundant system can withstand permanent corruptions of entries in the ith row(s) of $\mathcal{A}$ and/or $\mathcal{B}$.

[4] We use $A(i, l)$ to denote the element in the ith row and the lth column of matrix A.

TMR is clearly a special case of the above formulation: corruption of a state variable of the first subsystem is guaranteed to remain within the first subsystem. Therefore $M_f \subseteq \{1,2,...,n\}$ and (conservatively) $\Phi_a = \begin{bmatrix} 0 & I_n & I_n \end{bmatrix}^T$. One possible L_a is (among others) $\begin{bmatrix} 0 & I_n & 0 \end{bmatrix}$.

Less obvious is the following case (based on the earlier linear coding example). Consider the system with state evolution equation (8). Its decoding matrix is given by $L = \begin{bmatrix} I_4 & 0 \end{bmatrix}$. If $\mathcal{A}(2,2)$ (whose value is .5) becomes corrupted, then the set of indices of corrupted state variables is $M_{2_f} = \{2,5\}$. Below, we show the original encoding matrix Φ, together with the encoding matrix Φ_a (resulting after the corruption of entry $\mathcal{A}(2,2)$) and a suitable L_a:

$$\Phi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & 0 \\ -1 & -1 & 0 & -1 \\ -1 & 0 & -1 & -1 \end{bmatrix}, \quad \Phi_a = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & -1 \\ -1 & 0 & -1 & -1 \end{bmatrix},$$

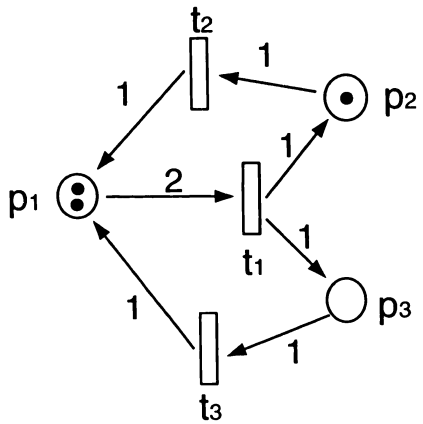
$$L_a = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Using the above L_a, the redundant system can continue to function properly (that is, provide the correct state vector $x[k]$ for all future time steps) despite the corrupted entry $\mathcal{A}(2,2)$. We can still use the parity check matrix of eq. (7) for fault detection, except that the checks involving the second and/or fifth state variables (i.e., the first and second checks in $\Theta\xi[k]$) are invalid.

3. REDUNDANT PETRI NETS

Petri nets are a graphical and mathematical model for a variety of information and processing systems, including concurrent, asynchronous, distributed, nondeterministic, and/or stochastic systems. They are particularly relevant to the study of discrete event systems (DES); theory, examples and applications can be found in [2, 7, 25, 30]. A Petri net is represented by a directed, bipartite graph with two kinds of nodes: *places* (denoted by $\{p_1, p_2, ..., p_n\}$ and drawn as circles) and *transitions* (denoted by $\{t_1, t_2, ..., t_m\}$ and drawn as rectangles). Weighted directed arcs connect transitions to places and vice-versa (but there are *no* connections from a place to a place or from a transition to a transition). The arc weights have to be non-negative integers (we use b_{ij}^- to denote the weight of the arc from place p_i to transition t_j and b_{lj}^+ to denote the weight of the arc from transition t_j to place p_l). The graph in Figure 1 is an example of a Petri net with three places and three transitions. By convention, arcs with zero weight are *not* drawn.

Places function as *token holders*. Tokens are drawn as black dots and can be regarded as representing resources that are available at different parts of a system. The number of tokens in a place cannot be negative. At any given time step k, the *marking* (state) of the Petri net is given by the number of tokens at its places; for the Petri net in the figure, the marking shown (say, at time step 0) is $x[0] = [\ 2 \quad 1 \quad 0\ ]^T$.



$$B = \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}$$

Fig. 1. A pure Petri net with 3 places and 3 transitions.

Transitions model *events* and cause the rearrangement or generation or disappearance of tokens. Transition t_j is *enabled* (i.e., it is allowed to take place) only if each of its input places p_i has at least b_{ij}^- tokens. When transition t_j takes place (we say that transition t_j *fires*), it removes b_{ij}^- tokens from each input place p_i, and adds b_{lj}^+ tokens to each output place p_l. In our example in Figure 1, transitions t_1 and t_2 are enabled, but transition t_3 is not. If transition t_1 fires, then it will remove 2 tokens from its input place p_1, and add 1 token each to its output places p_2 and p_3; the next state of the Petri net will be $x[1] = [\ 0 \quad 2 \quad 1\ ]^T$.

More generally, we can define $B^- = [b_{ij}^-]$ (respectively, $B^+ = [b_{ij}^+]$) to be the $n \times m$ matrix with b_{ij}^- (respectively, b_{ij}^+) at its ith row, jth column position. If we let $B \equiv B^+ - B^-$, the state evolution of a Petri net is represented by the following equation:

$$x[k+1] \;=\; x[k] + (B^+ - B^-)u[k] \equiv x[k] + Bu[k]\ . \tag{9}$$

(In Figure 1, we show the corresponding B for that Petri net.) The input vector $u[k]$ in the above description is restricted to have exactly one non-zero entry that is 1. When $u_j[k] = 1$, transition t_j fires (j in $\{1, 2, ..., m\}$). Of course, a transition cannot fire unless it is enabled.

A *pure* Petri net is one in which no place serves as both an input and an output for the same transition (i.e., only one of b_{ij}^+ and b_{ij}^- can be non-zero). The Petri net in Figure 1 is an example of a pure Petri net. Note that for a pure Petri net a

transition is enabled as long as the resulting state vector $x[k+1]$ (as given by eq. (9)) has non-negative integer entries. For the rest of this paper, we focus on pure Petri nets with initial conditions that make them *L1-live*, that is, given the initial state $x[0]$, each transition t_j in S can fire at least once under a particular firing sequence, [25]. The Petri net in Figure 1 with the indicated initial state is $L1$-live.

3.1. Systematic introduction of redundancy

One way of achieving fault tolerance in a pure Petri net is by *monitoring* the given set of transitions through additional places and tokens. We consider a fault-tolerant scheme in which a Petri net S (with n places) is embedded into a redundant Petri net $\mathcal{S}$ that has η places ($\eta \equiv n + d$, $d > 0$) and admits the same set of transitions as S. The state evolution of the redundant Petri net is given by

$$\xi[k+1] = \xi[k] + \mathcal{B}u[k] , \tag{10}$$

with the state vector $\xi[k]$ at time step k providing complete information about $x[k]$, the state of the original system S at k. Again, we require that there exist an $n \times \eta$ decoding matrix L and an $\eta \times n$ encoding matrix Φ, such that for all k, $x[k] = L\xi[k]$ and $\xi[k] = \Phi x[k]$. Since S and $\mathcal{S}$ are *pure* Petri nets, their state vectors (respectively, $x[k]$ and $\xi[k]$) should consist of non-negative integer entries and matrices B and $\mathcal{B}$ need to have integer entries. Additionally, in order for $\mathcal{S}$ to be an embedding of S, we need the set of transitions enabled in S to be a *subset* of the set of transitions enabled in $\mathcal{S}$ (i.e., we do not want the additional structure in $\mathcal{S}$ to inhibit any of the transitions allowed in S).

Theorem 2. Consider the setting described above. If the pure Petri net $\mathcal{S}$ (of dimension $\eta \equiv n + d$, $d > 0$) is an embedding of S, then its state evolution eq. (10) is *similar* (in the same sense as for LTI state-space systems) to a standard state evolution equation of the form

$$\xi_\sigma[k+1] = \xi_\sigma[k] + \begin{bmatrix} B \\ 0 \end{bmatrix} u[k] , \tag{11}$$

where B is the matrix in eq. (9), the state evolution of the original Petri net.

Proof. Follows from the LTI case. □

The standard state evolution eq. (11) corresponds to a pure Petri net $\mathcal{S}_\sigma$ which is trivially an embedding of S, because at any given time step, the transitions enabled in S and $\mathcal{S}_\sigma$ are the same. Associated with $\mathcal{S}_\sigma$ are the standard decoding matrix $L_\sigma = \begin{bmatrix} I_n & 0 \end{bmatrix}$, the standard encoding matrix $\Phi_\sigma = \begin{bmatrix} I_n \\ 0 \end{bmatrix}$, and the standard parity check matrix $\Theta_\sigma = \begin{bmatrix} 0 & I_d \end{bmatrix}$.

A few comments are in order regarding a converse to the theorem. If we start from the standard state evolution eq. (11) and the redundant system $\mathcal{S}_\sigma$, we can work our way back to a non-standard Petri net $\mathcal{S}$ by choosing an *appropriate* $\eta \times \eta$

transformation matrix $\mathcal{T}$. Given $\mathcal{T}$, the resulting system $\mathcal{S}$ has state vector $\xi[k] = \mathcal{T}\xi_\sigma[k]$ and state evolution $\xi[k+1] = \xi[k] + \mathcal{B}u[k]$, where $\mathcal{B} = \mathcal{T}^{-1}\begin{bmatrix} B \\ 0 \end{bmatrix} = \Phi B$. In order for $\mathcal{S}$ to be a valid Petri net description, we need $\mathcal{B} = \Phi B$ to have integer entries and $\xi[k] = \Phi x[k]$ to be a valid state vector for all valid $x[k]$ (a valid pure Petri net state vector is one that has non-negative integer entries)[5]. A sufficient condition is that the first n columns of $\mathcal{T}^{-1}$ consist of non-negative integer entries (this condition is necessary if all non-negative integer vectors $x[0]$ are allowable as initial conditions). This guarantees that the encoding matrix $\Phi = \mathcal{T}^{-1}\Phi_\sigma$ has non-negative integer entries (which in turn guarantees that $\mathcal{B} = \Phi B$ has integer entries and that, for any *valid* state $x[k]$ in S, the corresponding state $\xi[k] = \Phi x[k]$ in $\mathcal{S}$ is also valid). Note that other than $L\Phi = I_n$, no restrictions are placed on L. Specifically, the entries of L can be negative and/or fractional (as long as the constraints on Φ are satisfied).

3.2. Error model and examples

The errors expected in a Petri net model depend on the underlying system and the actual hardware implementation. One possibility is that each hardware failure affects the number of tokens that are transported/combined/processed in a *single* place of the Petri net. In terms of the state evolution equation, a single fault causes the value of a single state variable (in $x[k]$ or $\xi[k]$) to be incorrect. Such an error model is appropriate for Petri net models of finite state machines or linear automata, [33, 34] (because single-bit errors corrupt a single place of the Petri net). Other error models are also possible.

If we apply the similarity transformation $\xi[k] = \mathcal{T}\xi_\sigma[k]$, where $\mathcal{T}$ is given by $\begin{bmatrix} I_n & 0 \\ -C & I_d \end{bmatrix}$ and C is a $d \times n$ matrix with *non-negative* integer entries, we obtain (from $\mathcal{S}_\sigma$) a transformed system $\mathcal{S}$ that is a valid Petri net (because the first n columns of $\mathcal{T}^{-1}$ have non-negative entries). The encoding, decoding and parity check matrices are given by

$$\Phi = \mathcal{T}^{-1}\Phi_\sigma = \begin{bmatrix} I_n \\ C \end{bmatrix}, \quad L = L_\sigma\mathcal{T} = \begin{bmatrix} I_n & 0 \end{bmatrix}, \quad \Theta = \Theta_\sigma\mathcal{T} = \begin{bmatrix} -C & I_d \end{bmatrix}.$$

This construction was first suggested in [33] and emerges as a particular instance of our theorem. One uses the additional places to monitor the operation of the original system. They have no effect on the behavior of the original system because the transitions enabled in the two systems are the same.

The interpretation of the fault-tolerant scheme is straightforward: we add d places and connect them to the transitions (events) of the original Petri net. The weights of the additional connections are given by the entries in the matrix CB and enable us to monitor the flow of tokens by checking at each time step whether $\begin{bmatrix} -C & I_d \end{bmatrix}\xi[k]$

[5] This also ensures that the set of transitions enabled in S (and $\mathcal{S}_\sigma$) is a subset of the set of transitions enabled in $\mathcal{S}$. If t_j is enabled in S, then the next state vector $x[k+1] = x[k] + B(:, j)$ consists of non-negative (integer) entries. Since $\xi[k+1] = \Phi x[k+1]$, the state vector $\xi[k+1]$ is also non-negative, which means that transition t_j is enabled in $\mathcal{S}$ as well.

is zero. Note that, if C is chosen properly, we might be able to locate the place where the failure has occurred and the exact number of tokens that has been corrupted. For example, if $d = 1$ and $C = [\ 1 \quad 2 \quad 2\ ]$, then the pure Petri net of Figure 1 is protected through the use of one additional place as shown on the left side of Figure 1 (the additional connections are shown with dotted lines). The parity check is $\Theta\xi[k] = [\ -1 \quad -2 \quad -2 \quad 1\ ]\,\xi[k]$ and it is able to detect errors in a single state variable.

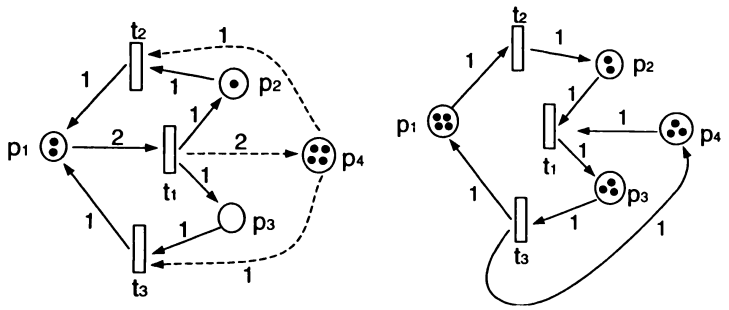


Fig. 2. Two different redundant versions of the pure Petri net in Figure 1.

Our scheme encompasses more general embeddings than in [33] by allowing us to restructure the original Petri net if necessary (thereby permitting fault tolerance considerations during the *design* of the overall Petri net). For example, the Petri net on the right side of Figure 2 is another redundant version of the Petri net of Figure 1. It uses one additional place ($d = 1$) and results from a more general transformation of the standard state evolution eq. (11). The parity check matrix is a checksum matrix of the form $\Theta = [\ -2 \quad -2 \quad 1 \quad 3\ ]$; the transformation matrix $\mathcal{T}^{-1}$ used, as well as matrix $\mathcal{B}$, and the encoding and decoding matrices are as follows:

$$\mathcal{T}^{-1} = \begin{bmatrix} 1 & 2 & 0 & -1 \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 2 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \\ -1 & 0 & 1 \end{bmatrix},$$

$$\Phi = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 0 \end{bmatrix}, \qquad L = \begin{bmatrix} 5 & 6 & -3 & -7 \\ -3 & -4 & 2 & 5 \\ -1 & -1 & 1 & 1 \end{bmatrix}.$$

Both fault-tolerant schemes in Figure 2 are able to detect errors in a single state variable by performing the checksum $\Theta\xi[k]$. If establishing connections is hard or if the transfer of tokens is expensive, however, then the approach on the right has some significant advantages: (i) it requires fewer connections between places and transitions (only 8 connections as opposed to 10 for the approach on the left), and (ii) the sum of weights is significantly less (only 8 as opposed to 12 for the scheme on the left).

4. CONCLUSION

We have outlined a systematic procedure for introducing structured redundancy into state-space systems. Our approach maps the state vector of the original system into a larger, redundant space, while ensuring that the initial conditions and evolution in the redundant space will preserve the state, evolution and properties of the original system. We have demonstrated through several examples how the added redundancy can be used for fault tolerance. Moreover, we have completely characterized all appropriate fault-tolerant designs for LTI state-space systems. In particular, we have illustrated that our method amounts to augmenting the original system with redundant modes that cannot be excited by the input and are initialized to zero; through appropriate design and hardware implementation, we can ensure that failures will excite these additional modes, thus allowing us to identify them and possibly correct all errors.

We have also given pointers on how to use a similar approach to study fault tolerance in Petri net models. Our future work will focus on further extending our results for Petri nets (e. g., by using different error models, by looking at distributed error detection and correction schemes, by investigating issues related to Petri net languages and supervisory control as in [12, 14], and by including unobservable/uncontrollable transitions as in [24]). We are also studying other classes of dynamic systems in state form, such as finite state machines and max-plus systems.

(Received April 8, 1998.)

REFERENCES

[1] J. A. Abraham: Fault tolerance techniques for highly parallel signal processing architectures. Proceedings of SPIE *614* (1986), 49–65.

[2] F. Baccelli, G. Cohen, G. J. Olsder and J. P. Quadrat: Synchronization and Linearity. Wiley, New York 1992.

[3] P. E. Beckmann: Fault-Tolerant Computation Using Algebraic Homomorphisms. Ph.D. Thesis. EECS Department, Massachusetts Institute of Technology, Cambridge, MA 1992.

[4] P. E. Beckmann and B. R. Musicus: A group-theoretic framework for fault-tolerant computation. In: IEEE Internat. Conf. on Acoustics, Speech, and Signal Processing, 1992, pp. 557–560.

[5] P. E. Beckmann and B. R. Musicus: Fast fault-tolerant digital convolution using a polynomial residue number system. IEEE Trans. Signal Process. *41* (1993), 2300–2313.

[6] J. W. Brewer, J. W. Bunce and F. S. Van Vleck: Linear Systems Over Commutative Rings. (Lecture Notes in Pure and Applied Mathematics 104.) Marcel Dekker, Inc., New York 1986.

[7] C. G. Cassandras: Discrete Event Systems. Aksen Associates, Boston 1993.

[8] C. G. Cassandras, S. Lafortune and G. J. Olsder: Trends in Control: A European Perspective. Springer-Verlag, London 1995.

[9] A. Chatterjee: Concurrent error detection in linear analog and switched-capacitor state variable systems using continuous checksums. In: Internat. Test Conference 1991, pp. 582–591.

[10] A. Chatterjee and M. d'Abreu: The design of fault-tolerant linear digital state variable systems: theory and techniques. IEEE Trans. Comput. *42* (1993), 794–808.

[11] V. Y. Fedorov and V. O. Chukanov: Analysis of the fault tolerance of complex systems by extensions of Petri nets. Automat. Remote Control *53* (1992), 2, 271–280.
[12] S. Gaubert and A. Giua: Deterministic weak–and–marked Petri net languages are regular. IEEE Trans. Automat. Control *AC-41* (1996), 12, 1802–1803.
[13] A. Ginzburg: Algebraic Theory of Automata: Academic Press, New York 1968.
[14] A. Giua and F. DiCesare: Decidability and closure properties of weak Petri net languages in supervisory control. IEEE Trans. Automat. Control *AC-40* (1995), 5, 906–910.
[15] C. N. Hadjicostis: Fault–Tolerant Computation in Semigroups and Semirings. M. Engr. Thesis. EECS Department, Massachusetts Institute of Technology, Cambridge, MA 1995.
[16] C. N. Hadjicostis and G. C. Verghese: Fault–tolerant computation in semigroups and semirings. In: Internat. Conf. on Digital Signal Processing, Vol. 2, Cyprus, 1995, pp. 779–784.
[17] C. N. Hadjicostis and G. C. Verghese: Fault–tolerant computation in groups and semigroups, submitted.
[18] K.-H. Huang and J. A. Abraham: Algorithm–based fault tolerance for matrix operations. IEEE Trans. Comput. *33* (1984), 518–528.
[19] M. Ikeda and D. D. Šiljak: An inclusion principle for dynamic systems. IEEE Trans. Automat. Control *AC-29* (1984), 3, 244–249.
[20] J.-Y. Jou and J. A. Abraham: Fault–tolerant matrix arithmetic and signal processing on highly concurrent parallel structures. Proc. IEEE *74* (1986), 732–741.
[21] J.-Y. Jou and J. A. Abraham: Fault–tolerant FFT networks. IEEE Trans. Comput. *37* (1988), 548–561.
[22] T. Kailath: Linear Systems. Prentice–Hall, Englewood Cliffs, NJ 1980.
[23] D. G. Luenberger: Introduction to Dynamic Systems: Theory, Models, & Applications. Wiley, New York 1979.
[24] J. O. Moody and P. J. Antsaklis: Supervisory control using computationally efficient linear techniques: a tutorial introduction. In: 5th IEEE Mediterranean Conf. on Control and Systems, Cyprus 1997.
[25] T. Murata: Petri nets: properties, analysis and applications. Proc. IEEE *77* (1989), 541–580.
[26] V. S. S. Nair and J. A. Abraham: Real–number codes for fault–tolerant matrix operations on processor arrays. IEEE Trans. Comput. *39* (1990), 426–435.
[27] J. P. Norton: Structural zeros in the modal matrix and its inverse. IEEE Trans. Automat. Control *AC-25* (1980), 980–981.
[28] A. V. Oppenheim and R. W. Schafer: Discrete–Time Signal Processing. Prentice Hall, Englewood Cliffs, NJ 1989.
[29] T. R. N. Rao: Error Coding for Arithmetic Processors. Academic Press, New York 1974.
[30] C. Reutenauer: The Mathematics of Petri Nets. Prentice Hall, New York 1990.
[31] R. A. Roberts and C. T. Mullis: Digital Signal Processing. Addison–Wesley, Reading, MA 1987.
[32] A. Sahraoui, H. Atabakhche, M. Courvoisier and R. Valette: Joining Petri nets and knowledge–based systems for monitoring purposes. In: IEEE Internat. Conf. Robotics Automation, Raleigh, NC 1987, pp. 1160–1165.
[33] J. Sifakis: Realization of fault–tolerant systems by coding Petri nets. J. Design Automation and Fault–Tolerant Computing *3* (1979), 2, 93–107.
[34] M. Silva and S. Velilla: Error detection and correction on Petri net models of discrete events control systems. In: Proceedings of the ISCAS 1985, pp. 921–924.
[35] J. von Neumann: Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. Princeton University Press, Princeton, NJ 1956.

[36] S. B. Wicker: Error Control Systems. Prentice Hall, Englewood Cliffs, NJ 1995.
[37] K. Yamalidou, J. Moody, M. Lemmon and P. Antsaklis: Feedback control of Petri net based on place invariants. Automatica *32* (1996), 1, 15–28.

Christoforos N. Hadjicostis, EECS Department, MIT, Cambridge, MA 02139. U. S. A.
e-mail: chadjic@allegro.mit.edu

George C. Verghese, EECS Department, MIT, Cambridge, MA 02139. U. S. A.
e-mail: verghese@mit.edu