# Algebraic Techniques for the Optimization of Control Flow Checking

Guevara NOUBIR and Berthe Y. CHOUEIRY
Department of Computer Science
Swiss Federal Institute of Technology in Lausanne (EPFL)
{guevara.noubir | berthe.choueiry}@di.epfl.ch

## Abstract

*In [4, 5], Leveugle addresses the problem of reducing the overhead of on-line testing in dedicated controllers. He introduces a low-overhead technique that allows the detection of illegal paths in finite state machines. Based on Leveugle's idea for detecting illegal paths, we introduce a* new simple *signature function. This signature function can be efficiently implemented in software. The assignment of values to the states is carried out algebraically by matrix inversion instead of using exhaustive search methods. We show that signatures computed using* `MISR` *or checksum are particular cases of our* more general *signature function. Thus, the state assignment problem defined in [4, 5] can be solved more efficiently.*

*Then, we address the problems of* latency *and* checking *from a formal perspective and show that finding the smallest set of checking states (i.e., states where the static signature is compared with the run-time signature) that induces a latency less than or equal to a given value L is* **NP***-hard and there exists no polynomial time algorithm that solves this problem unless* **P**=**NP***.*

## 1. Introduction

Techniques of control flow checking are widely used for supporting the reliability of computer systems. They can be applied in a system at various levels of abstraction (controller level, microprogram level, instruction level, application level, etc.). A survey of these techniques can be found in [4]. The basic principle of control flow checking is to verify that the run-time execution of a system (controller or program) corresponds to the expected/specified behavior. The expected behavior is given by the control flow graph of the system. The verification is done as follows: A signature function is used to compress the sequence of instructions or states that is currently being executed; then, this signature is compared to a static signature stored in the self-checking system. Various signature schemes are proposed in the literature [20, 7, 16, 19, 6, 5, 17] and are mainly implemented in hardware. The most popular signature functions are the checksum and the polynomial division by means of a multiple input shift register (`MISR`). In general, these techniques aim at improving one or more of the following parameters: error detection coverage, error detection latency, processor performance, memory overhead, and monitor complexity.

In [4], Leveugle addresses the problem of designing controllers with a control flow checking ability that induces a low overhead in a silicon implementation. The idea is to assign codes to the states such that this assignment verifies an invariant property. This property is that all paths leading to any given state have the same signature. Leveugle applies this technique to control flow graphs called SC-graphs, which are the graphs satisfying this property. He gives a procedure for recognizing such graphs and introduces rules to recast a non SC-graph into an SC-graph.

In this paper, we exploit, in a more general context, the idea proposed by Leveugle to reduce the complexity of the checking. Our contribution is three-fold:

1. We introduce a novel and generic algebraic signature function that can be efficiently implemented both in hardware and software.
2. We adapt the fault detection scheme proposed by Leveugle to software applications and provide new and efficient algorithms for implementing it.
3. We formalize the problem of latency checking and show that it is NP-hard.

This paper is organized as follows. In Section 2, we review the previous work in this area. In Section 3, we introduce our new signature function and show that classical signature functions are a special case of the one we propose. In Section 4, we provide algebraic solutions to the problem of assigning values to states. We show that this approach is more efficient

than exhaustive search and heuristic techniques used in [4, 2]. Finally, in Section 5, we formalize the problem of latency and examine its relationship to the complexity of the checker. We show that the problem of optimizing the checking is **NP**-hard.

## 2. Related work

The problem of control flow checking in a finite state machine (FSM) has also been addressed by Robinson and Shen [13, 14] and Escherman [2]. In [13], Robinson and Shen propose to replace the set of rules introduced by Leveugle (to transform a non SC-graph into an SC-graph) by efficient algorithms. Their technique, called graph repairing, is based on a state splitting operation. In [14], they propose a technique for assigning codes to states that guarantees that two distinct states have distinct codes. This technique allows to deduce the value of the entering signature of a state using the value of the state code. It allows near-zero latency without guaranteeing it because of the correlation between the value of the state code and that of the signature. The drawback of this technique is that it may require that length of a state code be as long as $2 \times \log$ (number of states).

In [2], Escherman proposes a graph repairing technique based on cutting transitions. In his scheme, the state register is implemented using a multi-functional test register (e.g., BILBO [3]). This state register/pattern generator can be activated at run-time to generate the next state value while the output of the circuit is only constrained to satisfy the unique signature condition. Thus, signature checking is not achieved on the actual state path, which may affect the efficiency of the fault detection procedure.

While previous work has concentrated on the improvement of the technique introduced by Leveugle for the hardware implementation of FSMs, we provide a generalization of this technique that is applicable to both software and hardware implementations. Although our contribution does not add to the power of the techniques proposed in the literature for hardware implementations, it provides an efficient and novel solution to software applications, such as control programs and communication protocols. Moreover, we show that some previously known results on `MISR` signatures can more easily be obtained with our polynomial-based formulation.

In fact, in this paper, we address the problem of control flow checking in FSMs from a broader and more general perspective than before. Our approach is broad because the signature function is a *generic* algebraic function that can be efficiently implemented both in hardware and software. It is general, because we show that classical signature functions, such as checksum and `MISR`, are a special case of the function we propose.

The use of our technique in software applications yields a fault detection mechanism that is more efficient than the techniques usually used for this purpose. For instance, the methods proposed in the literature for fault detection in communication protocols are generally based on a replication of the FSM of the protocol [12, 18, 1]. In contrast to this massive replication, the signature technique only requires an observer that is implemented as a one-dimensional table whose size is at most equal to the number of states in the FSM [11, 10]. Furthermore, a computation of the signature requires only one addition and one multiplication operations at each transition. An example of the use of this technique in a protocol application is shown in Appendix A.

## 3. A new signature scheme

The control flow graph is generally described by an FSM. We assume that the FSM is already in the form of an SC-graph. To insure this property, one can use the algorithms introduced by Robinson and Shen in [13]. In this section, after recalling some basic definitions [4], we introduce our signature function then show that three classical signature functions (namely: checksum, `XOR`, and `MISR`) are a special case of the one we propose.

In this paper, we adopt the following definitions:

**Finite State Machine (FSM)** . The system under observation is modeled as an FSM. An FSM is a 3-tuple $A = (Q, \Sigma, \delta)$; where $Q$ denotes the set of all possible states $\{S_1, \ldots, S_n\}$, $\Sigma$ is the set of all possible events $\{E_1, \ldots, E_m\}$, and $\delta$ denotes the state transition function ($\delta : Q \times \Sigma \rightarrow Q$).

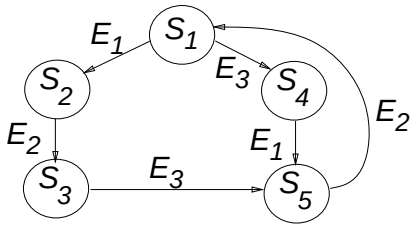


**Figure 1.** *A simple example of an FSM.*

A simple example of an FSM is shown in Fig. 1. Each state $S_i$ (respectively, event $E_i$) is associated with a value $s_i$ (respectively, $e_i$) from the working algebraic field $F$.

**Full Path.** A *full path* $P$ is defined as an alternate sequence of states and events.

**State path.** The *state path* is the sequence of states derived from the path by deleting all the events and retaining the states in their original order.

**Event path.** The *event path* is the sub-sequence derived from the path by deleting all the states and retaining all the events in their original order.

**Last.** `Last(`$P$`)` of a path $P$ is the last state in $P$.

**First.** `First(`$P$`)` of a path $P$ is the first state in $P$.

**Penultimate.** `Penultimate(`$P$`)` is the last but one state in a path.

**Length.** `Length(`$P$`)` is the length of a path $P$. It is equal to the number of transitions in $P$.

**Correctness.** A path $P$ is *correct* if and only if every sub-sequence $S_i\ E_j\ S_k$ of $P$ satisfies $\delta(S_i, E_j) = S_k$.

**Legality.** A state path $P = S_1\ S_2\ S_3\ \cdots\ S_n$ is a *legal state path* if and only if for every sub-sequence $S_iS_j$ of $P$, there exists an event $E \in \Sigma$ such that $\delta(S_i, E) = S_j$.

An event path $P = (E_1\ E_2\ E_3\ \cdots\ E_n)$ is a *legal event path* if and only if, for every sub-sequence $E_iE_j$ of $P$, there exists three states $(S_k, S_l, S_m) \in Q^3$ such that $\delta(S_k, E_i) = S_l$ and $\delta(S_l, E_j) = S_m$.

### 3.1. The polynomial signature function

In [11, 10], we propose three different signature functions based on the evaluation of a polynomial derived from the path. The functions that we propose exhibit the following characteristics: (1) They are stepwise computable (using Horner algorithm). (2) They are efficiently implementable (both in hardware and software). (3) They have a bounded aliasing probability. (4) They exhibit a structure suited for algebraic manipulations.

**Full path signature.** The `full path signature` function utilizes the state and event information to compute the signature. Thus, the states and the events must be visible to the signature generator. Each state or event is assigned a value that is used for computing the signature. The signature is computed by evaluating the path polynomial at a point $x_0$. This procedure detects incorrect paths and is well-suited for developing and implementing self-checking programs. The polynomial associated with a full path $P = (S_0\ E_0\ S_1\ E_1\ \ldots\ E_{n-1}\ S_n)$ is defined as follows:

$$P_P(x) = \sum_{i=0}^{n-1} (s_i x^{2(n-i)} + e_i x^{2(n-i)-1}) + s_n$$

where: $s_i$ is the state value; $e_i$ is the event value; $n$ is the length of the state path; and $x$ is a number from the working Galois field $F$ as discussed in Section 3.4. The full signature is defined as a function $\Phi$ that associates a number with a path $P$: $\Phi(P) = P_P(x_0)$.

**State signature.** The `state signature` is computed using only the state values. This technique is used when the generator has access to the current state information. This signature can detect illegal state paths but cannot detect illegal event paths. The polynomial associated with a state path $P = (S_0\ S_1\ \ldots\ S_n)$ is defined as follows:

$$P_P(x) = \sum_{i=0}^{n} s_i x^{n-i}$$

The state signature is: $\Phi_{state}(P) = P_P(x_0)$

**Event signature.** The `event signature` is computed using only the event values. This technique is used when the generator does not have access to the current state information but has access to that of the event. It is well-suited for detecting faults in communication protocols when the checker is an external observer [10]. In this case, the event values are the types of the frames that are exchanged during the execution of the protocol. The polynomial associated with an event path $P = (E_0\ E_1\ \ldots\ E_n)$ is defined as follows:

$$P_P(x) = \sum_{i=0}^{n} e_i x^{n-i}$$

The event signature is: $\Phi_{event}(P) = P_P(x_0)$

### 3.2. Aliasing probability

In [9], we prove that, when errors on bits are equally likely, the probability that two different sequences of states have the same signature is $\frac{1}{|F|}$. Furthermore, when those sequences differe in only one state, this probability is null.

### 3.3. Specialization of the polynomial signature

In this section, we show that the signatures computed by `XOR`'ing the state value, by checksum, and by `MISR` (with internal `XOR`) are in fact a special case of the signature function that we propose.

**`XOR` signatures.** When the working field is $GF(2^k)$ and $x_0 = 1$, then the resulting signature is equivalent to a `XOR` signature function.

**Checksum signatures.** When working in the field $GF(p)$ (where $p$ is a prime number) and when $x_0 = 1$, the signature function is equivalent to the arithmetic checksum signature used by [15].

**MISR signatures.** Signatures computed using internal XOR MISR circuits that have an irreducible compaction polynomial $g(x)$, see Fig. 2, are a special case of the signature function that we propose. The working field is an extension field $GF(2^k)$. This field is generated by extending $GF(2)$ using an irreducible polynomial $g(x)$ of degree $k$. $x_0$ is taken equal to $0 \cdots 010$, which we denote as $2 \in GF(2^k)$. The compaction polynomial of the MISR circuit is in fact the extension polynomial $g(x)$.

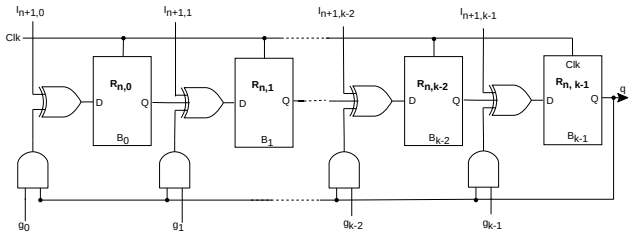


**Figure 2.** *An internal* XOR MISR *circuit.*

Let $R_{i,j}$ be the content of register $j$ at step $i$ and $I_{i,j}$ the $j^{th}$ bit of the $i^{th}$ input. $I_i$ and $R_j$ are considered as elements of the Galois field $GF(2^k)$. The MISR equation is as follows:

$$\begin{aligned} R_{n+1} &= [R_{n,k-2}, \ldots, R_{n,0}, 0] + [I_{n+1,k-1}, \ldots, I_{n+1,0}] \\ &\quad + R_{n,k-1}[g_{k-1}, \ldots, g_0] \\ R_{n+1} &= I_{n+1} + 2 \times R_n \end{aligned} \quad (1)$$

Addition and multiplication are done in $GF(2^k)$ (i.e., $+$ is equivalent to a XOR and $\times$ is a polynomial multiplication modulo the compaction polynomial $g(x)$). Thus,

$$\begin{aligned} R_n &= I_n + 2I_{n-1} + \ldots + 2^{n-1} I_1 \\ R_n &= \sum_{i=1}^{n} 2^{n-i} I_i \end{aligned} \quad (2)$$

This formulation shows that the MISR signature is a special case of the signature function that we propose in Section 3.1. Hence, the theory presented in this paper applies also to the classical MISR signature. Moreover, it is simpler than the classical MISR formulation and, because of its polynomial representation, it is better suited for algebraic manipulation.

### 3.4. Choice of $x_0$

When computing the signature of a path, the state (respectively, the event) values do not cover all the elements of the working Galois field. Consequently, the masking probability of the signature function may be sub-optimal. In order to obtain a good covering of the Galois field, one has to choose $x_0$ as a primitive root of unity. In the signature, the sum of the terms $x_0^i$ covers all the possible values of the field thus, yielding uniformly distributed signature values.

**Theorem 1** For MISR signatures, choosing $x_0$ as a primitive root of unity is equivalent to choosing a primitive compaction polynomial $g(x)$ for the MISR circuit.

**Proof:** Elements of $GF(2^k)$ can be interpreted as polynomials of degree $(k-1)$. Operations in this field are polynomial operations modulo the extension polynomial $g(z)$. The element 2 of $GF(2^k)$ is then represented as the polynomial $p(z) = z$. Taking $x_0 = 2$ is a primitive root of unity in $GF(2^k)$ is equivalent to:

$$\begin{aligned} &\forall\, d < 2^k - 1; && z^d \neq 1 \bmod g(z) \\ &\forall\, d < 2^k - 1 \text{ and } \forall\, k(z); && z^d - 1 \neq k(z) \cdot g(z) \end{aligned}$$

This means that $g(z)$ does not divide any polynomial $z^d - 1$ where $d < 2^k - 1$, which is the definition of a primitive polynomial. □

## 4. Three assignment problems

In [4, 5], Leveugle defines the state assignment problem as the problem of computing one value per state such that all correct paths leading to a given state are equal. Using our signature function, this problem can be generalized to three assignments problems. In this section, we introduce these problems, show how to solve them algebraically, and illustrate by an example. The three assignments problems are listed below.

**The state assignment problem.** Given an FSM $A = (Q, \Sigma, \delta)$, find the values to be associated with the states such that the *state* signatures of all correct paths leading to a given state are equal.

This kind of problems arises in the design of self-checking programs and controllers.

**The event assignment problem.** Given an FSM $A = (Q, \Sigma, \delta)$, find the values to be associated with the events such that the *event* signatures of all correct paths leading to a given state are equal.

This problem is relevant for event-driven systems and for external detection of errors in communication protocols.

**The state-event assignment problem.** Given an FSM $A = (Q, \Sigma, \delta)$, find the values to be associated with the state and events such that the *full* signatures of all correct paths leading to a given state are equal.

This problem is applicable to self-checking programs and communication protocols.

### 4.1. Solving the assignment problems

In this section, we show how to solve the state assignment problem. Algorithms for solving the event assignment problem and the state-event assignment problem are similar to the one described below and can be found in [9]. The state assignment problem is solved in two steps: First, it is transformed into a set of linear equations; then, the system of linear equations is solved by matrix inversion.

**Algorithm 1** *System of equations (input: A; output: System)*

$System \leftarrow \{\}$;
**If** $|S_0^-| > 0$ **Then**
  **For all** $S_i \in S_1^-$;
    **Let** $P_i$ a state path $|$ $(\texttt{First}(P_i) = S_0) \wedge (\texttt{Last}(P_i) = S_0) \wedge (\texttt{Penultimate}(P_i) = S_i)$;
    $eqn \leftarrow \Phi_{state}(P_i) = 0$;
    $System \leftarrow System \cup \{eqn\}$
**For all** $S \in Q - \{S_0\}$ $|$ $|S^-| > 1$
  **Let** $S_1 \in S^-$ and $P_0$ a state path $|$ $(\texttt{First}(P_0) = S_0) \wedge (\texttt{Last}(P_0) = S) \wedge (\texttt{Penultimate}(P_0) = S_1)$;
  **For all** $S_i \in S^- - \{S_1\}$
    **Let** $P_i$ : state path $|$ $(\texttt{First}(P_i) = S_0) \wedge (\texttt{Last}(P_i) = S) \wedge (\texttt{Penultimate}(P_i) = S_i)$;
    $eqn \leftarrow \Phi_{state}(P_0) = \Phi_{state}(P_i)$;
    $System \leftarrow System \cup \{eqn\}$
**end**

where $S^-$, the set of antecedents of $S$, denotes the set of states $S_i$ for which there exists a transition to $S$.

The first phase in the algorithm described above generates $|S_0^-|$ equations to guarantee that all paths leading to the initial state $S_0$ have the same signature. Since the initial state $S_0$ is assigned an arbitrary signature[1] 0, all paths leading to $S_0$ have a null signature. In the second phase, for every state $S$ different from $S_0$, the algorithm selects a state $S_1$ from $S^-$ and a path $P_0$ leading to $S$ through $S_1$. Then, for every state $S_i$ preceding $S$, the algorithm generates an equation stating that the signature of a path $P_i$ leading to $S$ through $S_i$ is equal to the signature of $P_0$.

Algorithm 1 generates $|T|-|Q|+1$ equations, where $|T|$ denotes the number of transitions in the FSM and $|Q|$ the number of states. This algorithm has a time complexity of $O(|T|)$. The system of linear equations generated as output can be written as a matrix. This matrix is generally not square. Therefore, in order to solve the system, one has to select an invertible submatrix of size $|T|-|Q|+1$. This operation implies that some states ($|T|-|Q|+1$) are part of the selected submatrix and some others ($2|Q|-|T|-1$) are left out. The latter are called *free states* and are chosen using the Algorithm 3 reported in Appendix B. These free states are in fact degrees of freedom, which can be exploited to reinforce other criteria (or constraints) such as to reduce the hardware complexity of a controller. The matrix inversion can be done using a Gauss elimination algorithm, which has a computational complexity of $O(n^3)$, where $n$ is the size of the matrix. When the number of equations (i.e., $|T| - |Q| + 1$) exceeds the number of states (i.e., $|Q|$), the state assignment problem cannot in general be solved unless one modifies the initial FSM [4, 13, 2]. In addition to this condition, the free states have to be chosen such that every state is assigned a distinct value.

In software implementations, the state codes can be chosen from a set of size much larger than $|Q|$. For instance, if the values of the free states are randomly chosen from a set[2] of size $|Q|^2$ (the values are coded over $2 \times \log(|Q|)$ bits), the probability of having the same code for two different states is less than $\frac{1}{2}$, [9]. Thus, a probabilistic algorithm that randomly chooses the values of the free states will find a solution in $k$ rounds with probability $1 - 2^{-k}$.

In hardware implementations, the state values have to be coded over a minimized number of bits. According to our formulation of the state assignment problem, the search is restricted only to the $2|Q| - |T| - 1$ free states, which yields a substantial reduction of the search space.

From the discussion above, it appears that solving the state assignment problem by using the algebraic properties of the signature function is more efficient than an exhaustive search carried over all the states of the FSM. It also appears that the checking operation of the FSM requires only a one-dimensional table instead of the two-dimensional one used for software applications modeled using an FSM [12, 18, 1].

### 4.2. Example

Consider the FSM presented in Fig. 3 left. When applied to this FSM, Algorithm 1 yields the system of equations described below:

$$\begin{cases} \Phi_{state}(S_1 a) = \Phi_{state}(S_1 b S_8 c) \\ \Phi_{state}(S_1 a S_2 a S_3 c S_4 d S_9 b) = \Phi_{state}(S_1 b S_8 d) \\ \Phi_{state}(S_1 a S_2 a S_3 e) = \Phi_{state}(S_1 a S_2 a S_3 c S_4 c S_5 a S_6 b) \\ \Phi_{state}(S_1 a S_2 a S_3 c S_4 c S_5 a S_6 b) \\ \qquad = \Phi_{state}(S_1 a S_2 a S_3 c S_4 d S_9 b S_{10} a) \\ \Phi_{state}(S_1 a S_2 a S_3 e S_7 f) = 0 \end{cases} \quad (3)$$

[1] Any arbitrary value can be chosen.

[2] The technique proposed in [14] may result in a coding of the states for a hardware implementation of the same length (i.e., $2 \times \log(|Q|)$ bits) as we require for the software implementation.

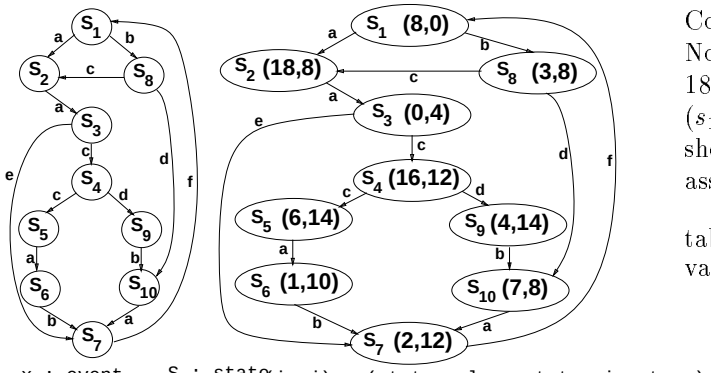


**Figure 3.** *Left:* the FSM. *Right:* the state coding.

Let $S$ be the state vector ($s_1$, $s_2$, $s_3$, $s_4$, $s_5$, $s_6$, $s_7$, $s_8$, $s_9$, $s_{10}$). The system in (3) becomes:

$$A \cdot S = 0 \tag{4}$$

$A$ being the matrix:

$$\begin{bmatrix} x_0-1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ {x_0}^4-x_0 & {x_0}^3 & {x_0}^2 & x_0 & 0 & 0 & 0 & -1 & 1 & 0 \\ {x_0}^5-{x_0}^2 & {x_0}^4-x_0 & {x_0}^3-1 & {x_0}^2 & x_0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_0 & 1 & 0 & 0 & -x_0 & -1 \\ {x_0}^3 & {x_0}^2 & x_0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

In order to solve this problem, we decompose matrix $A$ into two sub-matrices $A_1$ and $A_2$ such that $A_1$ is invertible. The states $S_6, S_7, S_3, S_9$ and $S_{10}$ are chosen as the free states using the Algorithm 3 in Appendix B.

$$A_1 \begin{pmatrix} s_1 \\ s_2 \\ s_8 \\ s_4 \\ s_5 \end{pmatrix} + A_2 \begin{pmatrix} s_6 \\ s_7 \\ s_3 \\ s_9 \\ s_{10} \end{pmatrix} = 0 \tag{5}$$

And,

$$A_1 = \begin{bmatrix} x_0-1 & 0 & 1 & 0 & 0 \\ {x_0}^4-x_0 & {x_0}^3 & -1 & x_0 & 0 \\ {x_0}^5-{x_0}^2 & {x_0}^4-x_0 & 0 & {x_0}^2 & x_0 \\ 0 & 0 & 0 & 0 & x_0 \\ {x_0}^3 & {x_0}^2 & 0 & 0 & 0 \end{bmatrix} \tag{6}$$

$$A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & {x_0}^2 & 1 & 0 \\ 1 & 0 & {x_0}^3-1 & 0 & 0 \\ 1 & 0 & 0 & -x_0 & -1 \\ 0 & 1 & x_0 & 0 & 0 \end{bmatrix} \tag{7}$$

The state values $s_1, s_2, s_8, s_4$ and $s_5$ can be computed from those of $s_6, s_7, s_3, s_9$ and $s_{10}$:

$$\begin{pmatrix} s_1 \\ s_2 \\ s_8 \\ s_4 \\ s_5 \end{pmatrix} = -A_1^{-1} A_2 \begin{pmatrix} s_6 \\ s_7 \\ s_3 \\ s_9 \\ s_{10} \end{pmatrix} \tag{8}$$

Consider the Galois field $GF(19)$ and let $x_0 = 3$. Note that $x_0$ is a primitive root of unity of order 18. Choosing $(s_6, s_7, s_3, s_9, s_{10}) = (1, 2, 0, 4, 7)$ yields $(s_1, s_2, s_8, s_4, s_5) = (8, 18, 3, 16, 6)$. In fig. 3 right, we show the resulting entering signature and the values associated with the states of the FSM.

The checker is implemented as one-dimensional table that contains, for every state, its code and the value of its entering signature, as shown below:

| State | Code | Sgn. | State | Code | Sgn. |
|---|---|---|---|---|---|
| $S_1$ | 8 | 0 | $S_6$ | 1 | 10 |
| $S_2$ | 18 | 8 | $S_7$ | 2 | 12 |
| $S_3$ | 0 | 4 | $S_8$ | 3 | 8 |
| $S_4$ | 16 | 12 | $S_9$ | 4 | 14 |
| $S_5$ | 6 | 14 | $S_{10}$ | 7 | 8 |

This one-dimensional table is smaller than the two-dimensional table of states and events used by a checker that replicates the initial FSM.

## 5. Checking optimization

In the previous sections, we showed how to reduce the complexity of the signature checking procedure: a checker needs only a one-dimensional table instead of the two-dimensional one otherwise required. In this section, we address the problem of further reducing the size of the one-dimensional table while guaranteeing a given delay for error detection.

The main idea can be summarized as follows. Let $Q_c$ be the set of states where the current signature is compared to the static signature of a state in the FSM. When $Q_c = Q$, this signature checking operation is carried out at *each* state of the FSM. Thus, the size of the one-dimensional signature table is equal to the size of $Q_c$. Restricting the checking operation to a subset of $Q$ yields a reduction of the size of the table while introducing some *latency* for fault-detection. The *fault-detection latency* associated with a set $Q_c$ of states, see Section 5.1, is defined as the maximum number of transitions contained in a correct path of which at most the root is in $Q_c$. Note that this problem is also of interest for the design of some high speed protocols that have to do periodically some state exchange [8].

First, we introduce an efficient algorithm for computing the latency associated with a given set $Q_c$. Then, we study the complexity of minimizing the size of $Q_c$ that guarantees a given latency and show that it is **NP**-hard.

### 5.1. Computing the fault-detection latency

We start with some formal definitions:

**Induced graph.** Given a graph $G(V, E)$ and a set of vertices $V' \in V$, the graph $G'(V', E')$ induced by

$V'$ is defined as follows: $\forall\, v_1, v_2 \in V'; (v_1, v_2) \in E' \Leftrightarrow \exists$ a path $P$ in $G \,|\, (\texttt{First}(P) = v_1) \wedge (\texttt{Last}(P) = v_2) \wedge (P - \{\texttt{First}(P), \texttt{Last}(P)\} \cap V' = \emptyset)$.

**Induced distance.** The induced distance $d_{G \setminus G'}$ by $V'$ on two vertices of $V'$ is equal to the maximal length of a path $P$ of $G$ containing no vertex of $V'$ except of the extremities: $d_{G \setminus G'}(v_1, v_2 \in V') = \texttt{Max}\{\texttt{Length}(P) \,|\, (\texttt{First}(P) = v_1) \wedge (\texttt{Last}(P) = v_2) \wedge (\{P - \{\texttt{First}(P), \texttt{Last}(P)\}\} \cap V' = \emptyset)\}$.

**Latency.** The latency of an induced graph $G'$ is defined as the maximum induced distance between two vertices of $V'$ minus 1: $\texttt{Latency}(G') = \texttt{Max}\{\, d_{G \setminus G'}(v_1, v_2) | (v_1, v_2) \in V' \,\} - 1$. Algorithm 2 computes the latency.

**Algorithm 2** *Latency* $(G(V, E), V')$

$D_0 = 0$;
$D_1[i] = max(D_0[i], max_{j \in V' | (i,j) \in E}\{1 + D_0[j]\})$;
n $\leftarrow$ 1;
**While** $(D_n \neq D_{n-1}) \wedge (n \leq |V|)$ **Do**
  $D_{n+1}[i] \leftarrow max(D_n[i], max_{j \in V \setminus V' | (i,j) \in E}\{1 + D_n[j]\})$;
  n$\leftarrow$ n+1;
**If** $D_n \neq D_{n-1}$ **Then Exit** "CYCLE"/*infinite latency */
**Else** *Latency* $\leftarrow max\{D_n[i]\} - 1$;
**Return** (*Latency*)

**end**

When the graph has no cycles, $D_n[i]$ gives the maximal length of a path $P$ connecting $v_i$ to a vertex in $V'$ such that at most the end-vertices of $P$ are in $V'$. Note that Algorithm 2 also detects cycles in the graph $G$, whenever the case arises. The worst-case time complexity of this algorithm $O(|V|^2)$.

### 5.2. Minimizing the size of $Q_c$

One interesting question is to study if it is possible to minimize the size of $Q_c$ that guarantees a given value $L$ for the latency. For this purpose, we introduce the following optimization problem (MinQc) and the decision problem (LCDP) associated with it.

**Minimizing** $Q_c$ (MinQc). Given an integer $L$ and a graph $G(V, E)$, find the smallest subset $V'$ of $V$ such that $V'$ induces a graph $G'(V' \in V, E')$ of latency $l$ less than or equal to $L$.

**Latency Checking Decision Problem** (LCDP). Given an integer $L$, a graph $G(V, E)$, and an integer $V_{max} \leq |V|$, is there a set $V'$ such that $V' \in V$, $|V'| \leq V_{max}$, and the latency of the graph $G'(V', E')$ induced by $V'$ is less than or equal to $L$?

In the following, we state some results related to the computational complexity of MinQc and of LCDP; then we outline the mathematical foundations of our claims. The detailed proofs can be found in [9].

**Lemma 1** The decision problem LCDP is in the class **NP**.

**Sketch of Proof:** Any set $V'$ that is a solution to the problem LCDP can be checked by verifying that the induced latency is less than or equal to $L$. This means that LCDP can be solved using a non-deterministic algorithm. Thus, the decision problem LCDP is in **NP**. □

**Lemma 2** The decision problem LCDP is **NP**-hard by reduction from 3SAT.

The proof of this lemma is sketched in Appendix C.

**Theorem 2** The decision problem LCDP is **NP**-complete.

**Proof:** The decision problem is both in the class of **NP** (see Lemma 1) and **NP**-hard (see Lemma 2), thus it is **NP**-complete. □

Since we prove that the decision problem is **NP**-complete, this implies that there is no polynomial time algorithm to solve the optimization problem unless **P=NP**. One way to address this difficult problem is to use a heuristic that *reduces* the size of $Q_c$ while ensuring a bounded latency $L$; note that the size of $Q_c$ is not guaranteed to be minimal. Such a heuristic, which still need further refinement, can be based on the following steps:

1. First, since a cycle that has no vertex in $Q_c$ causes the latency to be infinite, it appears clearly that at least one vertex in each cycle should be chosen to be in $Q_c$. This problem is also known to be difficult and a heuristic can be used for this purpose.
2. Then, starting from the set of vertices in $Q_c$, the next step is to break all directed paths that have a length greater than the maximum latency acceptable. This can be done by identifying the vertices that are not in $Q_c$ and that are at a distance[3] $L$ from a vertex in $Q_c$. These vertices[4] are put in $Q_c$. The process described in this step reiterates until no vertex in the graph is at a distance greater than $L$ from a node in $Q_c$.

[3] When the latency of the graph is greater than $L$, there always exists vertices in $G$ at distance $L$ because the weight of edges is one.

[4] One may choose to restrict oneself to only those vertices that are at distance $L$ from the largest number of vertices in $Q_C$.

## 6. Conclusion

In this paper, we introduced an efficient signature function based on polynomial evaluation. This signature function can be efficiently implemented in software. We showed that classical signature functions known in the literature (e.g., checksum, `MISR`) are a special case of the one we propose and that known results about these functions can be re-proven in a much simpler manner. We showed that the state assignment problem can be efficiently solved using an algebraic matrix inversion followed by a probabilistic algorithm for software applications or a reduced-space backtrack search for hardware implementations). Furthermore, we formalized the problem of optimizing the checking while ensuring a predefined latency. Finally, we studied to computational complexity of this problem and showed that it is **NP**-hard.

As hanging research issues, we propose the following improvements, which we are currently investigating:

- Define and evaluate heuristics for solving the optimization problem.
- State some of the criteria for the combinatorial optimization of the controller as equations to be solved algebraically.
- In software implementation, it may become possible to accept, for some given states, $k$ valid signatures instead of the unique one envisaged in this paper. In this case, the techniques we presented become applicable to a larger class of FSMs.


## Acknowledgments

The authors would like to thank Dr. Leveugle and the anonymous referees for their constructive remarks.


## Appendices

## A. Protocol application

The detection of execution errors in communication protocols has been addressed in the literature [12, 18, 1], where it is assumed that errors appear as incorrect transitions. In [11, 10], we have shown that detecting execution errors in communication protocols can be achieved by checking the signature of the execution path. The signature checker can be either (1) internal to the protocol or (2) external.

1. In the first case, the full path signature is used because the checker has access to both the state and event information.
2. In the second case, the checker has only access to the event information, consequently only the event path signature can be used. The signature checker consists of a one-dimensional table of size $|Q_c|$ and one-dimensional table containing the values of the events.

Fig. 4 shows the application of this technique to the FSM corresponding to a simplified version of the ISO transport protocol of class 4.

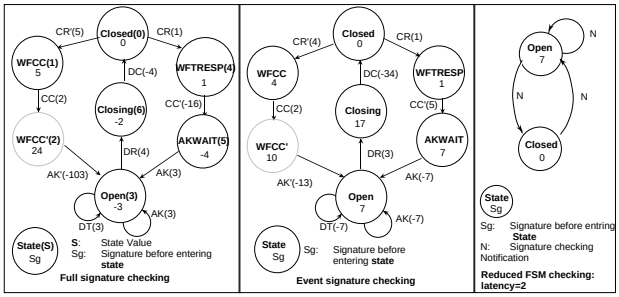


**Figure 4.** ISO transport protocol class 4. *Left:* A solution to the state-event assignment problem ($x_0 = 2$). *Center:* A solution to the event assignment problem. *Right:* Reduced FSM of TP4 with $Q_c = \{Open, Closed\}$ and $L = 2$.

## B. Selection of free states

The following algorithm takes as input an FSM $A$ and determines, as output, the set of free states discussed in Section 4.1. At each step, this algorithm selects an unconstrained state, then updates the set of states whose code (*Fixed-code*) or signature (*Fixed-sgn*), as a result of this decision, can no longer be freely chosen. The algorithm stops when no state whose code can be freely chosen is left (i.e., *Fixed-code* $= Q$), and it has a time complexity of $O(|T|)$; where $|T|$ denotes the number of transitions in the FSM and $|Q|$ the number of states. Note that $|\textit{Free-states}| = 2|Q| - |T| - 1$.

When this algorithm is used in the case of FSMs implemented in hardware, the complexity overhead of the hardware can be reduced by making an appropriate choice of the next state to be included in the set *Free-states*. For instance, the next state can be chosen among those that are adjacent to a state already in *Free-states* or they can be chosen from the set $Q_c$.

**Algorithm 3** *Selection of Free States (input: A; output: Free-states)*

```
Free-states ← {};
Fixed-code ← {};
Fixed-sgn ← {S0};
While Fixed-code ≠ Q Do
    Let S ∉ Fixed-code;
    Free-states ← Free-states ∪ {S};
    Succ ← Successors({S});
    While Succ ⊄ Fixed-sgn Do
        Fixed-sgn ← Fixed-sgn ∪ Succ;
        Pred ← Predecessors(Succ);
        Fixed-code ← Fixed-code ∪ Pred;
        Succ ← Successors(Pred)
end
```

## C. Proof of Lemma 2

For a given integer $L$, we show that it is possible to transform any instance of 3SAT into an instance of the problem LCDP. Let $C = \{c_1, \ldots, c_m\}$ be a set of *clauses* on the boolean *variables* $U = \{u_1, \ldots, u_n\}$. Each clause is formed of 3 *literals* and a literal is either a boolean variable $u_i$ or its negation $\overline{u_i}$. For instance, $c_x = (u_i \vee \overline{u_j} \vee \overline{u_k})$ is such a clause. The problem of 3SAT is the following: "is it possible to assign values (i.e., 0 or 1) to all the variables $u_i$ such that at least one literal in each clause is true?". In order to prove that LCDP is **NP**-hard, one has to:

1. define a mapping from an instance of a 3SAT problem into an instance of the LCDP problem,
2. prove that whenever the 3SAT instance can be satisfied, then the corresponding LCDP can be solved, and
3. prove that if the LCDP is soluble, then an assignment can be found that satisfies all the clauses of the 3SAT instance.

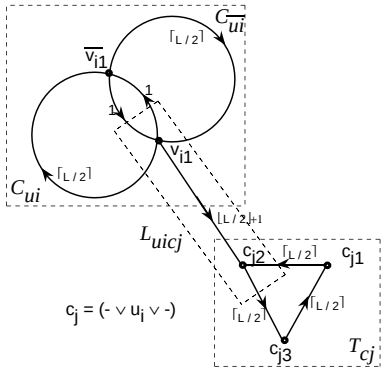


**Figure 5.** *Reduction:* Construction rules.

**Reduction.** We need to define a reduction from an instance of 3SAT into an instance of LCDP. First, we apply the following procedure to build a graph $G$ (Fig. 5 illustrates the steps of this procedure and Fig. 6 shows an example):

1. With each boolean variable $u_i$ we associate two cycles $C_{u_i}$ and $C_{\overline{u_i}}$, each of length $\lceil \frac{L}{2} \rceil + 1$. The subgraph induced by these two cycles is called the *double-cycle*[5] associated with the variable $u_i$. Each cycle is composed of vertices $v_{ix}$ and $\overline{v_{iy}}$ as follows:
   - $C_{u_i} = v_{i1} v_{i2} \cdots v_{i\lceil \frac{L}{2} \rceil} \overline{v_{i1}} v_{i1}$,
   - $C_{\overline{u_i}} = \overline{v_{i1}}\, \overline{v_{i2}} \cdots \overline{v_{i\lceil \frac{L}{2} \rceil}} v_{i1} \overline{v_{i1}}$.
2. With each clause $c_j$, we associate a triangle $T_{c_j} = (c_{j1}, c_{j2}, c_{j3})$ such that each literal of $c_j$ is associated with one of the vertices of the triangle $T_{c_j}$. The sides of $T_{c_j}$ have the same length, chosen equal to $\lceil \frac{L}{2} \rceil$, and are oriented to form a cycle.
3. The triangle $T_{c_j}$, which is associated with clause $c_j$, is then connected to each of the three double-cycles associated with the three literals of the clause $c_j$. The link is a path of length equal to $\lfloor \frac{L}{2} \rfloor + 1$ and oriented from the double-cycle towards the triangle. If $k^{th}$ $(1 \leq k \leq 3)$ literal of clause $c_j$ is:
   - a positive variable $u_i$, then the vertex $c_{jk}$ is linked to the vertex $v_{i1}$ in the double-cycle associated with $u_i$,
   - a negative variable $\overline{u_i}$, then the vertex $c_{jk}$ is linked to the vertex $\overline{v_{i1}}$ in the double-cycle associated with $u_i$.
4. The integer $V_{max}$ of the decision problem is taken equal to $n+2m$; where $n$ is the number of variables of the 3SAT instance and $m$ the number of clauses.

Note that the size of the graph generated by this procedure is linear in $(n, m)$.

[5] The expressions "double-cycle associated with a variable" and "double-cycle associated with a literal" are used interchangeably.

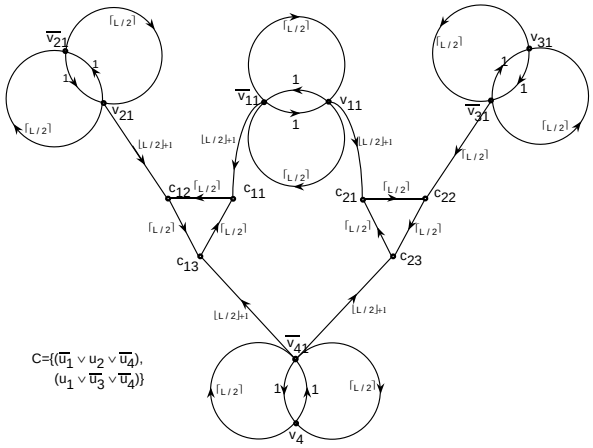


**Figure 6.** *Example:* a 3SAT instance reduced into a LCDP instance.

**Example.** In Fig. 6, we show the graph that corresponds to the following 3SAT instance:

$$C = \{(\overline{u_1} \vee u_2 \vee \overline{u_4}), (u_1 \vee \overline{u_3} \vee \overline{u_4})\}$$

**Sketch of Proof:** Below, we outline how the proof is constructed, the full proof can be found in [9].

- **The 3SAT instance is satisfiable $\Rightarrow$ $V'$ exists.** A clause $c_i$ is noted $(x_i \vee y_i \vee z_i)$. If the 3SAT instance is satisfiable, then one can built $V'$ as follows:

$$V' = \{v_{i1} | u_i = 1\} \cup \{\overline{v_{i1}} | u_i = 0\} \cup \left\{ \begin{array}{lll} \text{If } z_i = 1 & \text{Then } c_{i1}, c_{i2} & \\ & \text{ElseIf } y_i = 1 & \text{Then } c_{i1}, c_{i3} \\ & & \text{Else } c_{i2}, c_{i3} \end{array} \right\}$$

  The size of $V'$ is equal to $n + 2m$.

- **$V'$ exists $\Rightarrow$ the 3SAT instance is satisfiable.** The proof consists of showing that if there exists a set $V'$ such that every path $P$ of length $L + 1$ has at least one vertex (different from his first end-vertex) in $V'$, then we can find an assignment that satisfies all the clauses of the 3SAT instance.

  - In every double-cycle associated with a variable $u_i$, $\{v_{u_i}, \overline{v_{u_i}}\} \cap V' \neq \emptyset$ otherwise the latency induced by the cycle $v_{u_i}, \overline{v_{u_i}}, v_{u_i}$ would be infinite.
  - In every triangle, at least two vertices out of three are in $V'$.

  Therefore $|V'| \geq n + 2m$, but $|V'| \leq V_{max} = n + 2m$. Consequently, one and only one of $v_{u_1}$ and $\overline{v_{u_1}}$ is in $V'$; and only two vertices of each triangle associated with a clause are in $V'$. We conclude that every variable is assigned one and only one boolean value and every clause is satisfied by the literal associated with the triangle which vertex is not in $V'$. □

## References


[1] M. Diaz, G. Juanole, and J. Courtiat. Observer- A concept for Formal On-Line Validation of Distributed Systems. *IEEE Trans. on Soft. Eng.*, 20(12):900–912, 1994.

[2] B. Escherman. On Combining Off-Line BIST and On-Line Control Flow Checking. In *FTCS'22*, 298–305, Boston, MA, 1992.

[3] B. Könemann, J. Mucha, and G. Zwiehoff. Built-in Logic Block Observation Techniques. In *ITC*, 37–41, 1979.

[4] R. Leveugle. *Analyse de Signature et Test en Ligne Integré sur Silicium.* PhD thesis, Institut National Polytechnique de Grenoble, 1990.

[5] R. Leveugle and G. Saucier. Optimized Synthesis of Concurrently Checked Controllers. *IEEE Trans. on Comp.*, 39:419–425, 1990.

[6] A. Mahmood and E. J. McCluskey. Concurrent Error Detection Using Watchdog Processors - survey. *IEEE Trans. on Comp.*, 37(2):160–174, 1988.

[7] M. Namjoo. Techniques for Concurrent Testing of VLSI Processor Operation. In *ITC*, 461–468, 1982.

[8] A. N. Netravali, W. D. Roome, and K. Sabnani. Design and Implementation of a High-Speed Transport Protocol. *IEEE Trans. on Comm.*, 38:2010–2022, 1990.

[9] G. Noubir. *Nouvelles Techniques pour la Tolérance aux Pannes Basées sur l'Algèbre des Polynômes.* PhD thesis, Swiss Federal Inst. of Tech. in Lausanne (EPFL), 1996. In preparation.

[10] G. Noubir, K. Vijayananda, and H. J. Nussbaumer. A Robust Transport Protocol for Run-Time Fault Detection. In *Proc. ICNP'95*, 164–171, Tokyo, Japan, 1995.

[11] G. Noubir, K. Vijayananda, and P. Raja. Signature Based Technique for Fault Detection in Communication Protocols. In *Proc. of ISIT*, page 43, Whistler, Canada, 1995.

[12] M. Riese. *Model-Based Diagnosis of Communication Protocols.* PhD thesis, Swiss Federal Inst. of Tech., Lausanne, 1993.

[13] S. H. Robinson and J. P. Shen. Evaluation and Synthesis of Self-Monitoring State Machines. In *ICCAD'90*, 276–279, 1990.

[14] S. H. Robinson and J. P. Shen. Direct Methods for Synthesis of Self-Monitoring State Machines. In *FTCS'22*, 306–315, Boston, MA, 1992.

[15] N. R. Saxena and E. J. McCluskey. Control-Flow Checking using Watchdog assist and Extended-Precision Checksums. In *FTCS'19*, 1989.

[16] J. P. Shen and M. A. Schuette. On-line Self-monitoring using Signatured Instruction Streams. In *ITC*, 275–282, 1983.

[17] S. J. Upadhyaya and B. Ramamurthy. Concurrent Process Monitoring with No Reference Signatures. *IEEE Trans. on Comp.*, 43:475–480, 1994.

[18] C. Wang and M. Schwartz. Fault Detection with Multiple Observers. *IEEE Trans. on Networking*, 1(1):48–55, 1993.

[19] K. D. Wilken and J. P. Shen. Embedded Signature Monitoring: Analysis and Technique. In *ITC*, 324–333, 1987.

[20] S. S. Yau and F.-C. Chen. An Approach to Concurrent Control Flow Checking. *IEEE Trans. on Soft. Eng.*, 6:126–137, 1980.