Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems

Marcos V. Moreira, Thiago C. Jesus, and João C. Basilio

***Abstract*—The first step in the diagnosis of failure occurrences in discrete event systems is the verification of the system diagnosability. Several works have addressed this problem using either diagnosers or verifiers for both centralized and decentralized architectures. In this technical note, we propose a new algorithm to verify decentralized diagnosability of discrete event systems. The proposed algorithm requires polynomial time in the number of states and events of the system and has lower computational complexity than all other methods found in the literature. In addition, it can also be applied to the centralized case.**

***Index Terms*—Automata, computational complexity, diagnosability verification, discrete-event systems (DES), failure diagnosis.**

I. Introduction

Failure diagnosis is an important task in large complex systems and, as such, has received considerable attention [1]–[12]. In [4] and [5], a discrete-event system (DES) approach to failure diagnosis has been presented and a diagnoser has been proposed with two purposes: (*i*) on-line detection and isolation of system failures and; (*ii*) off-line verification of the diagnosability properties of the system. As shown in [4], on-line detection of failure events can be carried out with polynomial complexity at each step of the diagnosis procedure. However, the verification of language diagnosability, as proposed in [4], has, in the worst case, exponential complexity in the system state-space. This is due to the fact that the necessary and sufficient condition for diagnosability is stated in terms of the off-line diagnoser, and the state space of the diagnoser is, in the worst case, exponential in the cardinality of the state space of the system model.

More efficient ways to test whether or not a language is diagnosable—according to the language diagnosability definition introduced in [4]—without having to construct a diagnoser, have been presented in [13] and [14], in which polynomial-time algorithms, based on the construction of nondeterministic automata and on the search for cycles with a given property, are proposed. The computational complexity of the algorithm proposed in [13] is shown to be of fourth order in the number of states and of first order in the number of events of the system whereas the computational complexity of the algorithm proposed in [14] is of second order in the number of states and of first order in the number of events of the system; having, therefore, lower computational complexity than that proposed in [13]. It is important to remark that in [13] and [14] it is assumed that the language generated by the system is live and that there does not exist cycles of unobservable events in the automaton model of the system.

Polynomial time algorithms for the verification of diagnosability in the case of decentralized failure diagnosis (or codiagnosability) have been proposed in [15]–[17]. Decentralized diagnosis (or codiagnosis) implies that several diagnosers are deployed, each one possessing its own set of sensors, without any communication between them or to any coordinator; this is equivalent to protocol 3 of [18]. These algorithms are based on the construction of testing automata and on the search for offending cycles in these automata. The algorithm proposed in [15] can also be used for the verification of diagnosability and has computational complexity of order $(m+1)$ in the number of states and events of the system, where m denotes the number of local diagnosers. In the centralized case, $m = 1$, and the complexity of the algorithm is of second order in the number of states and events of the system model, which is greater than the order of complexity of the algorithm proposed in [14]. On the other hand, in [15] the assumptions of liveness of the language generated by the system and nonexistence of cycles of unobservable events are both removed. The algorithm proposed in [16] also removes these assumptions and the verifier automaton has $2^{m+1} \times |X|^{m+1}$ states and $2^{m+1} \times |X|^{m+1} \times |\Sigma| \times (m+1)$ transitions at most, where X and Σ are the state-space and event set of the system, respectively, and $|.|$ denotes cardinality.[1]

We propose in this technical note a new algorithm for verification of both codiagnosability and centralized diagnosability of DES and show that the proposed algorithm has lower computational complexity than the aforementioned ones. The algorithm efficiency is due to the fact that, in the construction of the verifier, only the traces that lead to violation of codiagnosability are actually searched, and as a by-product, theses traces can be easily found. We also remove, as in [15] and [16], the assumptions of liveness of the language generated by the system and nonexistence of cycles of unobservable events.

This technical note is organized as follows. Section II presents some preliminary concepts and reviews the definition of codiagnosability. In Section III, a new algorithm to test the decentralized and centralized diagnosability of discrete event systems is proposed. The computational complexity of the algorithm is evaluated in Section IV. In Section V, an example is used to illustrate the method. Finally, conclusions are drawn in Section VI.

II. Preliminaries

Let $G = (X, \Sigma, \Gamma, f, X_m, x_0)$ denote the deterministic automaton model of a DES, where X is the finite state space, Σ is the set of events, Γ is the feasible event function, f is the transition function, X_m is the set of marked states and x_0 is the initial state of the system. For the sake of simplicity, the feasible event function and the set of marked states are omitted unless stated otherwise [19]–[21].

The accessible part of G, denoted as $Ac(G)$, is the operation over G that eliminates all states of G that are not reachable from the initial state x_0 and its related transitions, i.e., $Ac(G) = (X_{ac}, \Sigma, f_{ac}, x_0)$, where $X_{ac} = \{x \in X : (\exists s \in \Sigma^\star)[f(x_0, s) = x]\}$ and $f_{ac} : X_{ac} \times \Sigma \to X_{ac}$ is the new transition function obtained by restricting the domain of f to the smaller domain of the accessible states X_{ac}. The coaccessible part of G, denoted as $CoAc(G)$, is obtained by eliminating all states of G from which it is not possible to reach a marked state, i.e., $CoAc(G) = (X_{coac}, \Sigma, f_{coac}, x_{0,coac}, X_m)$, where $X_{coac} = \{x \in X : (\exists s \in \Sigma^\star)[f(x, s) \in X_m]\}$, $x_{0,coac} = x_0$ if $x_0 \in X_{coac}$, or $x_{0,coac}$ is undefined if $x_0 \notin X_{coac}$, and $f_{coac} : X_{coac} \times \Sigma \to X_{coac}$ is the new transition function obtained by restricting the domain of f to the coaccessible states X_{coac}.

The projection $P_s : \Sigma_l^\star \to \Sigma_s^\star$, where $\Sigma_s \subset \Sigma_l$ is defined as $P_s(\epsilon) = \epsilon$, $P_s(\sigma) = \sigma$, if $\sigma \in \Sigma_s$ or $P_s(\sigma) = \epsilon$, if $\sigma \in \Sigma_l \setminus \Sigma_s$, and $P_s(s\sigma) = P_s(s)P_s(\sigma)$, for all $s \in \Sigma_l^\star$, and $\sigma \in \Sigma_l$. The projection operation can also be applied to the language generated by G,

Manuscript received June 23, 2010; revised December 01, 2010 and February 08, 2011; accepted February 28, 2011. Date of publication March 10, 2011; date of current version July 07, 2011. This work was supported in part by the Brazilian Research Council (CNPq). Recommended by Associate Editor H. Marchand.

The authors are with the COPPE-Programa de Engenharia Elétrica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ 21949-900, Brazil (e-mail: moreira@dee.ufrj.br; thiagocj@ufrj.br; basilio@dee.ufrj.br).

Digital Object Identifier 10.1109/TAC.2011.2124950

[1]The notation $|.|$ is also used in this technical note to denote the length of a sequence.

$\mathcal{L}(G)$, simply by applying these rules to all traces $s \in \mathcal{L}(G)$. The inverse projection $P_s^{-1} : \Sigma_s^\star \to 2^{\Sigma_l^\star}$ is defined as $P_s^{-1}(t) = \{s \in \Sigma_l^\star : P_s(s) = t\}$. The inverse projection operation can also be applied to $\mathcal{L}(G)$ to obtain $P_s^{-1}(\mathcal{L}(G))$. With a slight abuse of notation, the automaton that generates $P_s^{-1}(\mathcal{L}(G))$ will be denoted as $P_s^{-1}(G)$.

Operations between two or more automata can also be defined. Let $G_1 = (X_1, \Sigma_1, \Gamma_1, f_1, x_{0,1})$ and $G_2 = (X_2, \Sigma_2, \Gamma_2, f_2, x_{0,2})$. The product or completely synchronous composition of G_1 and G_2, denoted by $G_1 \times G_2$, is defined as $G_1 \times G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\times 2}, (x_{0,1}, x_{0,2}))$, where $f_{1\times 2}((x_1, x_2), \sigma) = (f_1(x_1, \sigma), f_2(x_2, \sigma))$, if $\sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2)$ and is undefined, otherwise. Notice that a transition in the product $G_1 \times G_2$ occurs if and only if the transition is possible in both automata G_1 and G_2. As a consequence, the language generated by $G_1 \times G_2$ is equal to $\mathcal{L}(G_1) \cap \mathcal{L}(G_2)$. The synchronous or parallel composition is defined as $G_1 \| G_2 = P_1^{-1}(G_1) \times P_2^{-1}(G_2)$, where $P_i : (\Sigma_1 \cup \Sigma_2)^\star \to \Sigma_i^\star$, $i = 1, 2$.

Suppose that the event set of G is partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o and Σ_{uo} denote the set of observable and unobservable events, respectively, and let $\Sigma_f \subseteq \Sigma_{uo}$ denote the set of failure events. In addition, assume that the set of failure events can be partitioned as

$$\Sigma_f = \bigcup_{i=1}^{\ell} \Sigma_{fi} \tag{1}$$

where Σ_{fi} represents a set of failure events of the same type. Let the language generated by G be denoted as $\mathcal{L}(G) = L$. The diagnosis of a failure event belonging to the set Σ_{fi} can be carried out using two basic architectures: centralized or decentralized. In the centralized architecture, a failure event in the set Σ_{fi} is diagnosed if it can be identified through the observation of events of the set Σ_o only. In the decentralized architecture, event observations are distributed among m local diagnosers, where each diagnoser has its own set of observable events and do not communicate between each other or to any coordinator. As a consequence, the occurrence of a failure event in the set Σ_{fi} is diagnosed when at least one local diagnoser identifies its occurrence. For this reason, the problem of diagnosing all failure event occurrences in a decentralized architecture as above is usually referred to as codiagnosis. When there are multiple failure types, then the diagnosability/codiagnosability of L with respect to Σ_{fi} is checked by constructing a verifier for Σ_{fi} treating all the other failure types as normal unobservable events. Thus, in this technical note, without loss of generality, it is considered that there is only one failure type, i.e., $\ell = 1$.

Suppose that there are m local diagnosers, each one with observable event set Σ_{o_i}, $i = 1, \ldots, m$, and assume, without loss of generality, that L is live. If L is not live, then there must be a deadlock state in the system, and so, L can be made live by adding a self-loop labeled with an unobservable event $\sigma_u \in \Sigma_{uo} \setminus \Sigma_f$ (the set of all nonfailure events that are not observed by any of the m local diagnosers) to each deadlock state. The resulting language will be live and such that all traces in L have the same projections $P_{o_i} : \Sigma^\star \to \Sigma_{o_i}^\star$ as before, therefore, not changing the partial observability property of the system. This procedure is similar to that described in [15] with the only difference that, in [15], the unobservable event is the empty trace ϵ. Since, in this technical note, automaton G is supposed to be deterministic, the self-loop added here is an unobservable event σ_u.

Let G_N be the subautomaton of G that represents the nonfailure behavior of the system, and let $\mathcal{L}(G_N) = L_N$; thus L_N is a prefix-closed language formed with all traces of L that do not contain any failure event from the set Σ_f.

Definition 1: Let L be the prefix-closed language generated by the system and let $L_N \subset L$ denote the prefix-closed language generated by G_N. Assume there are m local sites with projections $P_{o_i} : \Sigma^\star \to \Sigma_{o_i}^\star$ ($i \in I_M = \{1, \ldots, m\}$) and let Σ_f be the set of failure events. Then, L is codiagnosable with respect to P_{o_i} and Σ_f if and only if

$$(\exists n \in \mathbb{N})(\forall s \in L \setminus L_N)\,(\forall st \in L \setminus L_N, |t| \geq n) \Rightarrow$$
$$(\exists i \in I_M)\left(\forall w \in P_{o_i}^{-1}\left(P_{o_i}(st)\right) \cap L, w \in L \setminus L_N\right).$$

□

According to Definition 1, L is codiagnosable with respect to P_{o_i} and Σ_f if and only if for all traces st of arbitrarily long length after the occurrence of a failure event, there do not exist traces $s_i \in L_N$, not necessarily distinct, such that $P_{o_i}(s_i) = P_{o_i}(st)$ for all $i \in I_M$. Therefore, the verification algorithm is based on the search for traces $s_i \in L_N$, for $i = 1, \ldots, m$, and $st \in L \setminus L_N$ that violate the codiagnosability condition presented in Definition 1.

III. Verification Algorithm

In this section, we first present an algorithm for the verification of codiagnosability of a system and, in the sequel, we state a theorem that proves the algorithm correctness. It will be assumed, without loss of generality, that $m = 2$. The overall strategy behind the algorithm is simple but the presentation of detailed steps is somewhat involved; the reader may prefer to get insight into the proposed algorithm by consulting Example 1 of Section V while following the algorithm.

Algorithm 1: Let G be a deterministic automaton and Σ_f the set of failure events. Assume that, for each local diagnoser, Σ is partitioned as $\Sigma = \Sigma_{o_i} \dot{\cup} \Sigma_{uo_i}$, $i = 1, 2$, where Σ_{o_i} and Σ_{uo_i} are the observable and unobservable event sets for each local diagnoser, respectively.

- Step 1: Compute automaton G_N that models the normal behavior of G, as follows:
  - —Step 1.1: Define $\Sigma_N = \Sigma \setminus \Sigma_f$.
  - —Step 1.2: Build automaton A_N composed of a single state N (also its initial state) with a self-loop labeled with all events in Σ_N.
  - —Step 1.3: Construct the nonfailure automaton $G_N = G \times A_N = (X_N, \Sigma, f_N, \Gamma_N, x_{0,N})$.
  - —Step 1.4: Redefine the event set of G_N as Σ_N, i.e., $G_N = (X_N, \Sigma_N, f_N, \Gamma_N, x_{0,N})$.
- Step 2: Compute automaton G_F that models the failure behavior of the system, as follows:
  - —Step 2.1: Define $A_l = (X_l, \Sigma_f, f_l, x_{0,l})$, where $X_l = \{N, F\}$, $x_{0,l} = \{N\}$, $f_l(N, \sigma_f) = F$ and $f_l(F, \sigma_f) = F$, for all $\sigma_f \in \Sigma_f$.
  - —Step 2.2: Compute $G_l = G \| A_l$ and mark all states of G_l whose second coordinate is equal to F.
  - —Step 2.3: Compute the failure automaton[2] $G_F = CoAc(G_l)$.
- Step 3: Define function $R_i : \Sigma_N \to \Sigma_{R_i}$ as:[3]

$$R_i(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_{o_i} \\ \sigma_{R_i}, & \text{if } \sigma \in \Sigma_{uo_i} \setminus \Sigma_f \end{cases}. \tag{2}$$

  Construct automata $G_{N,i} = (X_N, \Sigma_{R_i}, f_{N,i}, x_{0,N})$, for $i = 1, 2$, with $f_{N,i}(x_N, R_i(\sigma)) = f_N(x_N, \sigma)$ for all $\sigma \in \Sigma_N$.
- Step 4: Compute the verifier automaton $G_V = G_{N,1} \| G_{N,2} \| G_F = (X_V, \Sigma_{R_1} \cup \Sigma_{R_2} \cup \Sigma, f_V, x_{0,V})$.

[2]Notice that all traces of G that contain a failure event belong to the language generated by G_F, i.e., $\mathcal{L}(G_F) = \overline{L \setminus L_N}$, where for a given language L defined over the event set Σ, $\bar{L}$ denotes the prefix-closure of L, i.e., $\bar{L} = \{u \in \Sigma^\star : (\exists v \in \Sigma^\star)[uv \in L]\}$.

[3]Notice that function R_i just renames the labels of the events in $\Sigma_{uo_i} \setminus \Sigma_f$. The notation $R_i(\Sigma_N)$ is used in this technical note to represent the renaming of the events in Σ_N as given by (2); thus, $\Sigma_{R_i} = R_i(\Sigma_N)$.

Notice that a state of G_V is given by $x_V = (x_{N,1}, x_{N,2}, x_F)$, where $x_{N,1}$, $x_{N,2}$, and x_F are states of $G_{N,1}$, $G_{N,2}$, and G_F, respectively, and $x_F = (x, x_l)$, where x and x_l are states of G and A_l, respectively.

- Step 5: Verify the existence of a cycle[4] $cl := (x_V^k, \sigma_k, x_V^{k+1}, \ldots, x_V^l, \sigma_l, x_V^k)$, where $l \geq k > 0$, in G_V satisfying the following conditions:

$$\exists j \in \{k, k+1, \ldots, l\} \; s.t. \text{ for some } x_V^j, \left(x_l^j = F\right) \wedge (\sigma_j \in \Sigma).$$

If the answer is yes, then L is not codiagnosable with respect to P_{o_i} and Σ_f. Otherwise, L is codiagnosable. □

It is important to remark that the verification algorithm proposed in this technical note has a crucial difference from that presented in [15] as follows: the search carried out in step 5 of Algorithm 1 seeks sequences of L_N that have the same projections as some sequence of $\mathcal{L}(G_F) = \overline{L \setminus L_N}$, whereas the search on the verifier proposed in [15] is for sequences of L_N that have the same projections as some sequence of L. As a consequence, the number of transitions of the verifier proposed here in comparison with that of [15] is significantly smaller, therefore reducing the complexity order of the algorithm. Finally, it is worth mentioning that Algorithm 1 only deploys parallel composition and does not require any new automaton composition as is the case of the algorithm presented in [15].

The following theorem proves the correctness of Algorithm 1.

Theorem 1: Let L and L_N ($L_N \subset L$) denote the prefix-closed languages generated, respectively, by G and G_N. Assume there are two local sites with projections $P_{o_i} : \Sigma^\star \to \Sigma_{o_i}^\star$ ($i = 1, 2$) and let Σ_f be the set of failure events. Then, L is not codiagnosable with respect to P_{o_i} and Σ_f if and only if there exists a cycle in G_V, $cl := (x_V^k, \sigma_k, x_V^{k+1}, \ldots, x_V^l, \sigma_l, x_V^k)$, where $l \geq k > 0$, satisfying the following conditions:

$$\exists j \in \{k, k+1, \ldots, l\} \; s.t. \text{ for some } x_V^j, \left(x_l^j = F\right) \wedge (\sigma_j \in \Sigma). \quad (3)$$

Proof: ($\Leftarrow$) Suppose that there exists a cycle $cl := (x_V^k, \sigma_k, x_V^{k+1}, \ldots, x_V^l, \sigma_l, x_V^k)$ satisfying condition (3). Since $x_l^j = F$ for some $j \in \{k, k+1, \ldots, l\}$, then, from the construction of A_l and G_V, it can be seen that $x_l^j = F$ for all $j \in \{k, k+1, \ldots, l\}$. This implies that there exists an event sequence $s_V t_V \in \mathcal{L}(G_V)$, such that $\sigma_f \in s_V$, where $\sigma_f \in \Sigma_f$, and $t_V = (\sigma_k \sigma_{k+1} \ldots \sigma_l)^p$, $p \in \mathbb{N}$, where $|t_V| > n, \forall n \in \mathbb{N}$. Define now the following projection operations:

$$P_{R_1} : (\Sigma \cup \Sigma_{R_1} \cup \Sigma_{R_2})^\star \to \Sigma_{R_1}^\star,$$
$$P_{R_2} : (\Sigma \cup \Sigma_{R_1} \cup \Sigma_{R_2})^\star \to \Sigma_{R_2}^\star,$$
$$P : (\Sigma \cup \Sigma_{R_1} \cup \Sigma_{R_2})^\star \to \Sigma^\star.$$

Since $G_V = G_{N,1} \| G_{N,2} \| G_F$, then $\mathcal{L}(G_V) = P_{R_1}^{-1}[\mathcal{L}(G_{N,1})] \cap P_{R_2}^{-1}[\mathcal{L}(G_{N,2})] \cap P^{-1}[\mathcal{L}(G_F)]$, which implies that $s_V t_V \in P^{-1}[\mathcal{L}(G_F)]$. Let $st = P(s_V t_V)$, where $s = P(s_V)$ and $t = P(t_V)$. Thus, since $P[P^{-1}(\mathcal{L}(G_F))] = \mathcal{L}(G_F)$, then $st \in \mathcal{L}(G_F)$. Notice that since $t_V = (\sigma_k \sigma_{k+1} \ldots \sigma_l)^p$, where $|t_V| > n, \forall n \in \mathbb{N}$, and, by assumption, there exists an event $\sigma_j \in \Sigma$ for $j \in \{k, k+1, \ldots, l\}$, then the event sequence $t = P(t_V)$ also has arbitrarily long length, which implies that $st \in L$ also has arbitrarily long length after the occurrence of the failure event σ_f.

Let $s_{R_1} = P_{R_1}(s_V t_V)$. Since $s_V t_V \in \mathcal{L}(G_V)$, then $s_V t_V \in P_{R_1}^{-1}[\mathcal{L}(G_{N,1})]$. In addition, $P_{R_1}[P_{R_1}^{-1}(\mathcal{L}(G_{N,1}))] = \mathcal{L}(G_{N,1})$, which implies that $s_{R_1} \in \mathcal{L}(G_{N,1})$. Notice that $G_{N,1}$ is obtained from G_N after renaming the events of the set Σ_N according to function R_1. Thus, there exists a trace $s_1 \in \mathcal{L}(G_N)$ such that $P_{o_1}(s_1) = P(s_{R_1})$. Using the same reasoning it can be shown that there exist traces $s_{R_2} \in \mathcal{L}(G_{N,2})$ and $s_2 \in \mathcal{L}(G_N)$ such that $P_{o_2}(s_2) = P(s_{R_2})$.

To conclude the proof, notice that

$$P(s_{R_1}) = P[P_{R_1}(s_V t_V)] = P_{R_1}[P(s_V t_V)] = P_{R_1}(st)$$

and thus $P_{o_1}(s_1) = P_{R_1}(st)$. Since $st \in \mathcal{L}(G_F) \subseteq \Sigma^\star$, $P_{R_1}(st) = P_{o_1}(st)$ yielding

$$P_{o_1}(s_1) = P_{o_1}(st).$$

The same reasoning can be used to show that $P_{o_2}(s_2) = P_{o_2}(st)$. These facts show that there exist traces $s_1, s_2 \in L_N$ and $st \in L \setminus L_N$ that violate the codiagnosability condition of Definition 1.

($\Rightarrow$) Suppose that L is not codiagnosable with respect to P_{o_i} and Σ_f. Thus, there exist sequences $st \in \mathcal{L}(G_F)$, where $\sigma_f \in s$ and $|t| > n, \forall n \in \mathbb{N}$, and $s_1, s_2 \in \mathcal{L}(G_N)$, such that $P_{o_1}(st) = P_{o_1}(s_1)$ and $P_{o_2}(st) = P_{o_2}(s_2)$. We will show that there exists a cycle of failure states in G_V associated with traces st, s_1 and s_2, that violates the codiagnosability condition of Definition 1 and, for this purpose, we split the proof in two parts, as follows: (*i*) in the first part we show that there exists an arbitrarily long length sequence $v \in \mathcal{L}(G_V)$ such that $P(v) = st$, $P_{R_1}(v) = s_{R_1}$, and $P_{R_2}(v) = s_{R_2}$, where $s_{R_1} = R_1(s_1)$ and $s_{R_2} = R_2(s_2)$;[5] (*ii*) in the second part we prove that there exists a cycle cl, associated with sequence v, satisfying condition (3).

In order to prove part (*i*), suppose that there exists a state in G_V, $x_V = (x_{N,1}, x_{N,2}, x_F)$, reachable from the initial state $x_{0,V}$ after the execution of a sequence $u \in \mathcal{L}(G_V)$, where $P(u)$ is in the prefix-closure of $\{st\}$, i.e., $P(u) \in \overline{\{st\}}$. Notice that this state x_V always exists since u can be the empty string and, in such case, $x_V = x_{0,V}$. Now, let $\sigma \in \Sigma$ be a feasible event of x_F, such that $P(u)\sigma \in \overline{\{st\}}$, and consider the problem of finding a state of G_V, $\hat{x}_V$, reachable from x_V, that has σ as a feasible event. Three cases are possible: (*a*) σ is observable by only one local diagnoser; (*b*) σ is observable by both local diagnosers; (*c*) σ is an unobservable event, i.e., $\sigma \in \Sigma_{uo}$.

Let us first consider case (*a*) and suppose, for instance, that $\sigma \in \Sigma_{o_1} \setminus \Sigma_{o_2}$. Therefore, in order to perform $G_V = G_{N,1} \| G_{N,2} \| G_F$, a self-loop labeled with σ must be introduced in all states of $G_{N,2}$. Thus, σ can occur if and only if it is a feasible event of the state associated with $G_{N,1}$. Since $P_{o_1}(s_1) = P_{o_1}(st)$, it can be seen that, after the occurrence of a finite trace of unobservable events in $\Sigma_{R_1}^\star$, a state of $G_{N,1}(\hat{x}_{N,1})$ that has σ as a feasible event must be reached. When this state is reached, σ will be a feasible event of $\hat{x}_V = (\hat{x}_{N,1}, x_{N,2}, x_F)$ as desired. Consider now case (*b*), i.e., $\sigma \in \Sigma_{o_1} \cap \Sigma_{o_2}$. In this case, σ will be a feasible event of x_V if and only if it is feasible for the corresponding states of $G_{N,1}$ and $G_{N,2}$. Since $P_{o_1}(s_1) = P_{o_1}(st)$ and $P_{o_2}(s_2) = P_{o_2}(st)$, then σ will be feasible for a state of G_V, $\hat{x}_V = (\hat{x}_{N,1}, \hat{x}_{N,2}, x_F)$, after the occurrence of a finite trace from $(\Sigma_{R_1} \cup \Sigma_{R_2})^\star$. Finally, consider case (*c*), i.e., $\sigma \in \Sigma_{uo}$. In this case, a self-loop labeled with all events in the set Σ_{uo} is added to each state of $G_{N,1}$ and $G_{N,2}$, which implies that σ is already feasible for $x_V = (x_{N,1}, x_{N,2}, x_F)$. Therefore, it can be seen that there exists a sequence v associated with st such that $v \in P_{R_1}^{-1}(s_{R_1}) \cap P_{R_2}^{-1}(s_{R_2}) \cap P^{-1}(st)$, which implies that $P(v) = st$, $P_{R_1}(v) = s_{R_1}$, and $P_{R_2}(v) = s_{R_2}$.

In order to prove part (*ii*), i.e., that there exists a cycle cl in G_V, associated with v, where at least one of the events in the cycle belongs to Σ, notice that since G_F is a finite state automaton, then associated

[4]A path $(x_k, \sigma_1, x_{k+1}, \sigma_2, \ldots, \sigma_l, x_{k+l})$, for $l > 0$, is the sequence of states and events such that $x_{k+i} = f(x_{k+i-1}, \sigma_i)$ for all $i \in \{1, 2, \ldots, l\}$. The path forms a cycle if $x_{k+l} = x_k$.

[5]The extension of function R_i to domain $\Sigma^\star$ is done in the usual way, i.e., $R_i(s\sigma) = R_i(s)R_i(\sigma)$, for all $s \in \Sigma^\star$ and $\sigma \in \Sigma$, and $R_i(\epsilon) = \epsilon$.

with st there is a cycle of failure states cl_F in G_F that can be associated with a path in G_V where the events of cl_F are contained in this path. Since G_V is a finite state automaton and $st = P(v)$, this path must include a cycle cl where at least one of the events in cl belongs to Σ. Moreover, since cl_F is a cycle of failure states, it can be seen from the construction of the verifier automaton G_V that cl is a cycle of failure states satisfying condition (3). ■

Remark 1: The proof of theorem 1 provides an easy way to find the traces s_1, s_2, and st that lead to a violation of codiagnosability of L, as follows:

1) Identify a cycle that satisfies the condition for non-codiagnosability imposed in step 5 of Algorithm 1 and obtain a trace v that takes $x_{0,V}$ to this cycle.
2) Obtain $s_{R_1} = P_{R_1}(v)$, $s_{R_2} = P_{R_2}(v)$ and $st = P(v)$.
3) Define the inverse renaming function

$$R_i^{-1} : \Sigma_{R_i} \to \Sigma_N$$
$$\sigma_{R_i} \mapsto \sigma$$

where $\sigma_{R_i} = R_i(\sigma)$, with the following extension to domain $\Sigma_{R_i}^{\star}$: $R_i^{-1}(s_{R_i}\sigma_{R_i}) = R_i^{-1}(s_{R_i})R_i^{-1}(\sigma_{R_i})$ for all $s_{R_i} \in \Sigma_{R_i}^{\star}$ and $\sigma_{R_i} \in \Sigma_{R_i}$, and $R_i^{-1}(\epsilon) = \epsilon$.

4) Obtain $s_1 = R_1^{-1}(s_{R_1})$ and $s_2 = R_2^{-1}(s_{R_2})$. □

IV. Complexity Analysis of Algorithm 1

The computational complexity of Algorithm 1 will be determined based solely on the analysis of the steps necessary to obtain the verifier automaton G_V, since the verification of the existence of offending cycles in G_V, required in Step 5 of Algorithm 1, can be carried out simply by decomposing the directed graph that results from the application of Algorithm 1 into strongly connected components [22]. According to the algorithm, if one vertex of a strongly connected component has an F label, then so do all vertices. Therefore, it is not necessary to count all the offending cycles in G_V but only to verify if there exists an edge labeled with an event in Σ connecting vertices labeled with F of a strongly connected component. As a result, Step 5 requires linear complexity in the number of states and transitions of G_V [22].

Table I shows the maximum number of states and transitions of all automata that must be computed in order to obtain the verifier automaton G_V for m local diagnosers according to Algorithm 1. The first step of Algorithm 1 is the construction of a single-state automaton A_N and the computation of the nonfailure automaton $G_N = G \times A_N$. Therefore, since A_N is a single-state automaton with a self-loop transition labeled with $\Sigma_N = \Sigma \setminus \Sigma_f$, then the maximum number of states and transitions of G_N are $|X|$ and $|\Sigma| - |\Sigma_f|$, respectively. The second step of Algorithm 1 is the construction of automaton G_F. In order to do so, it is first necessary to construct automaton A_l with two states, N and F, whose transitions are labeled only with failure events and, in the sequel, to obtain $G_l = G \| A_l$. Notice that $\mathcal{L}(G_l) = \mathcal{L}(G)$ and that the states of G_l are equal to (x, N) or (x, F), where $x \in X$. Therefore, the maximum number of states of G_l is $2 \times |X|$. The failure automaton G_F is computed by taking the coaccessible part of G_l whose set of marked states is composed of states of the form (x, F), $x \in X$. This leads to an automaton that generates language $\mathcal{L}(G_F) = \overline{L \setminus L_N}$. Since $G_F = CoAc(G_l)$, then, in the worst case, both automata have the same number of states and transitions. In step 3 automata $G_{N,i}$, for $i = 1, \ldots, m$, are obtained from G_N by renaming the unobservable events of the set $\Sigma_{uo_i} \setminus \Sigma_f$ for each diagnoser according to function R_i defined in (2). This leads to m automata with the same number of states and transitions of G_N. Finally, in step 4 the verifier automaton G_V is obtained by computing $G_V = G_{N,1} \| G_{N,2} \| \ldots \| G_{N,m} \| G_F$.

TABLE I
Computational Complexity of Algorithm 1

	No. of states	No. of transitions
G	$\|X\|$	$\|X\| \times \|\Sigma\|$
A_N	1	$\|\Sigma\| - \|\Sigma_f\|$
G_N	$\|X\|$	$\|X\| \times (\|\Sigma\| - \|\Sigma_f\|)$
A_l	2	$2\|\Sigma_f\|$
G_l	$2\|X\|$	$2\|X\| \times \|\Sigma\|$
G_F	$2\|X\|$	$2\|X\| \times \|\Sigma\|$
$G_{N,i}$	$\|X\|$	$\|X\| \times (\|\Sigma\| - \|\Sigma_f\|)$
G_V	$2\|X\|^{m+1}$	$2\|X\|^{m+1} \times [\|\Sigma\| + m(\|\Sigma\| - \|\Sigma_f\|)]$
Complexity		$O(m \times \|X\|^{m+1} \times (\|\Sigma\| - \|\Sigma_f\|))$

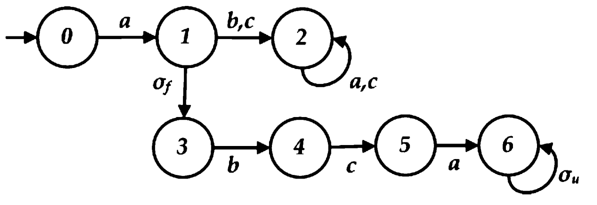

Fig. 1. Automaton G of Example 1.

Since the number of states of $G_{N,i}$ and G_F are, respectively, $|X|$ and $2 \times |X|$ at most, then the number of states of G_V is in the worst case equal to $2 \times |X|^{m+1}$. Moreover, the maximum number of transitions of G_V is equal to $2 \times |X|^{m+1} \times [|\Sigma| + m \times (|\Sigma| - |\Sigma_f|)]$, since for the construction of each $G_{N,i}$, $|\Sigma| - |\Sigma_f|$ new events can be created. Therefore, the complexity of Algorithm 1 is $O(m \times |X|^{m+1} \times (|\Sigma| - |\Sigma_f|))$, which shows that the proposed algorithm requires polynomial time in the number of states and events of G. It is worth remarking that, like all methods found in the literature, it has exponential complexity in the number of local diagnosers.

Notice that the computational complexity of Algorithm 1 is smaller than the complexities of the algorithms proposed in [15] and [16] which are $O(|X|^{m+1} \times |\Sigma|^{m+1})$ and $O(m \times 2^m \times |X|^{m+1} \times |\Sigma|)$, respectively. It is also important to remark that the size of the verifier automaton G_V is, in general, smaller than the worst case presented in Table I since the algorithm searches only for those traces in $L \setminus L_N$ and L_N that may lead to a violation of diagnosability, namely the verifier automaton represents only the traces in $\overline{L \setminus L_N}$ and L_N that have same projections.

Remark 2: It is important to remark that a test for the diagnosability of L with respect to a projection P_o and set of failure events Σ_f in the centralized case can be easily obtained by making $m = 1$ in Algorithm 1. Therefore, a verifier automaton for the centralized case is given as $G_{V_c} = G_{N,1} \| G_F$ and the necessary and sufficient condition for the nondiagnosability of L is the existence of a cycle of failure states in G_{V_c} such that at least one event in the cycle is an event of Σ. □

V. Example

Example 1: Consider the system modeled by automaton G shown in Fig. 1, and suppose we want to verify the codiagnosability of L with respect to P_{o_i}, $i = 1, 2$ and Σ_f, where $\Sigma = \{a, b, c, \sigma_u, \sigma_f\}$, $\Sigma_{o_1} = \{a, b\}$, $\Sigma_{o_2} = \{a, c\}$, $\Sigma_{uo} = \{\sigma_u, \sigma_f\}$, and $\Sigma_f = \{\sigma_f\}$. According to Algorithm 1, the first step is to obtain single-state automaton A_N and to compute the nonfailure automaton G_N, which are shown in Figs. 2(a) and 2(b), respectively. The next step is to obtain automaton G_l, shown in Fig. 3(b), by making the synchronous composition of G and A_l (shown in Fig. 3(a)) and marking all states that has F as the second component. Continuing Algorithm 1, we must construct the failure automaton G_F, depicted in Fig. 4,

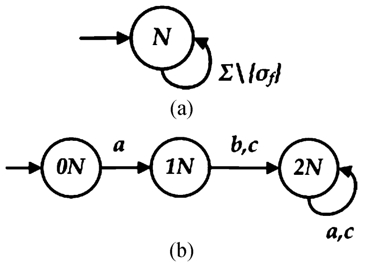
Fig. 2. Automata A_N (a) and G_N (b) of Example 1.

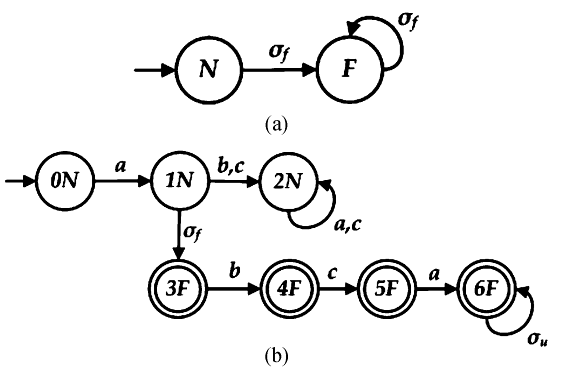
Fig. 3. Automata A_l (a) and $G_l = G||A_l$ (b) of Example 1.

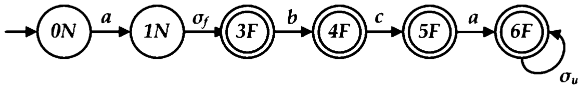
Fig. 4. Automaton $G_F = CoAc(G_l)$ of Example 1.

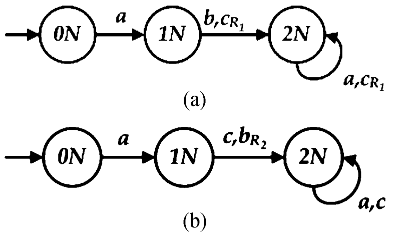
Fig. 5. Automata $G_{N,1}$ (a) and $G_{N,2}$ (b) of Example 1.

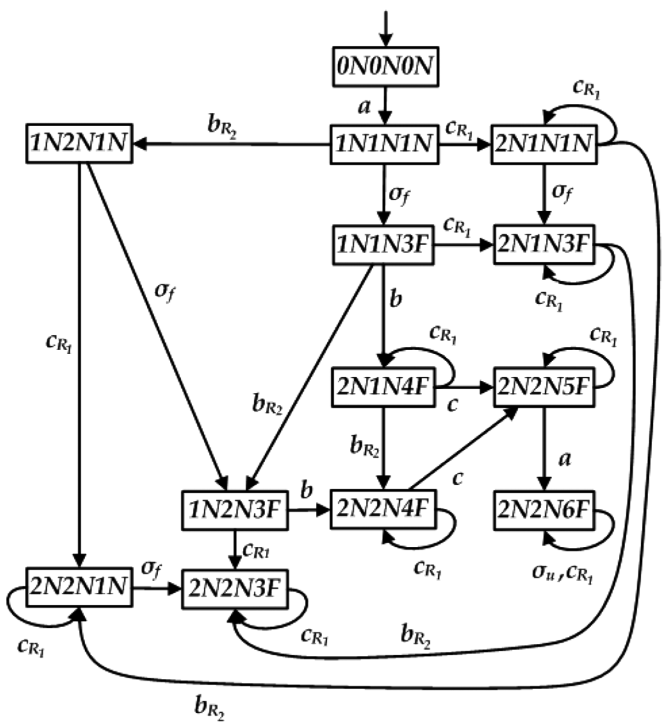
Fig. 6. Verifier automaton $G_V = G_{N,1}||G_{N,2}||G_F$ of Example 1.

taking the coaccessible part of G_l. The next step of Algorithm 1, is to obtain automata $G_{N,1}$ and $G_{N,2}$ (shown in Figs. 5(a) and 5(b)) from G_N by renaming the unobservable events in the sets $\Sigma_{uo_1} \setminus \Sigma_f = \{c, \sigma_u\}$ and $\Sigma_{uo_2} \setminus \Sigma_f = \{b, \sigma_u\}$, respectively. The final step of Algorithm 1 is the computation of the verifier automaton $G_V = G_{N,1}||G_{N,2}||G_F$, depicted in Fig. 6. To verify the codiagnosability it is necessary to find cycles of failure states in G_V formed with events in Σ. The verifier of Fig. 6 has several cycles (for example, the cycles $(2N1N3F, c_{R_1}, 2N1N3F)$, $(2N2N5F, c_{R_1}, 2N2N5F)$, and $(2N2N6F, \sigma_u, 2N2N6F)$). Notice that only cycle $(2N2N6F, \sigma_u, 2N2N6F)$ has events in Σ (all the other have only events in either $\Sigma_{R_1} \setminus \Sigma_f$ or $\Sigma_{R_2} \setminus \Sigma_f$). The existence of cycle $(2N2N6F, \sigma_u, 2N2N6F)$ implies that the language generated by G is not codiagnosable with respect to P_{o_i} and Σ_f.

As pointed out in remark 1, traces s_1, s_2, and st that lead to a violation of codiagnosability of L can be obtained from traces of G_V that contain cycles that violate the codiagnosability condition of theorem 1. It can be seen from Fig. 6 that these traces are: $v' = a\sigma_f bc^l_{R_1} cc^m_{R_1} ac^n_{R_1}\sigma^p_u$; $v'' = a\sigma_f b_{R_2} bc^l_{R_1} cc^m_{R_1} ac^n_{R_1}\sigma^p_u$; and $v''' = ab_{R_2}\sigma_f bc^l_{R_1} cc^m_{R_1} ac^n_{R_1}\sigma^p_u$, where $l, m, n, p \in \mathbb{N}$. Thus, the traces that lead to a violation of the codiagnosability are the failure trace $st = P(v') = P(v'') = P(v''') = a\sigma_f bca\sigma^p_u$ and the nonfailure trace $s_1 = R_1^{-1}(P_{R_1}(v')) = R_1^{-1}(P_{R_1}(v'')) = R_1^{-1}(P_{R_1}(v''')) = abc^l c^m ac^n$, with respect to P_{o_1}, and the nonfailure traces $s'_2 = R_2^{-1}(P_{R_2}(v')) = aca$ or $s''_2 = R_2^{-1}(P_{R_2}(v'')) = R_2^{-1}(P_{R_2}(v''')) = abca$, with respect to P_{o_2}.

Regarding the computational complexity, the number of states and transitions of the verifier automaton of Fig. 6 are, respectively, 13 and 28, whereas the number of states and transitions of the testing automaton proposed in [15] for automaton G of Fig. 1 are 16 and 68, respectively, and the number of states and transitions of the verifier automaton proposed in [16] are 69 and 168, respectively. As expected the search for the cycle that violates the codiagnosability has higher computational complexity using the verifiers proposed in [15] and [16].

VI. Conclusion

We propose in this technical note a new algorithm for the verification of decentralized and centralized diagnosability of a discrete event system using a deterministic verifier. This algorithm requires polynomial time in the cardinalities of the state space and event set of the system and has lower computational complexity than other algorithms proposed in the literature. The algorithm also does not require assumptions on the liveness of the language generated by the system or the nonexistence of cycles of unobservable events.

Acknowledgment

The authors would like to thank the anonymous reviewers for their useful comments and suggestions.

REFERENCES

[1] N. Viswanadham and T. L. Johnson, "Fault detection and diagnosis of automated manufacturing systems," in *Proc. 27th Conf. Decis. Control*, Austin, TX, 1988, pp. 2301–2306.

[2] P. Frank, "Fault diagnosis in dynamic systems using analytical knowledge based redundancy—A survey and new results," *Automatica*, vol. 26, pp. 459–474, 1990.

[3] F. Lin, "Diagnosability of discrete event systems and its applications," *J. Discrete Event Dyn. Syst.*, vol. 4, no. 2, pp. 197–212, 1994.

[4] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Trans. Autom. Control*, vol. 40, no. 9, pp. 1555–1575, Sep. 1995.

[5] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Failure diagnosis using discrete-event models," *IEEE Trans. Control Syst. Technol.*, vol. 4, no. 2, pp. 105–124, Mar. 1996.

[6] R. K. Boel and J. H. van Schuppen, "Decentralized failure diagnosis for discrete-event systems with costly communication between diagnosers," in *Proc. Int. Workshop Discrete Event Syst.*, Zaragoza, Spain, 2002, pp. 175–181.

[7] S. Tripakis, "Fault diagnosis for timed automata," in *Lecture Notes in Computer Sciences, In Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT)*. New York: Springer-Verlag, 2002, vol. 2469.

[8] S. H. Zad, R. H. Kwong, and W. M. Wonham, "Fault diagnosis in discrete-event systems: Framework and model reduction," *IEEE Trans. Autom. Control*, vol. 48, no. 7, pp. 1199–1212, Jul. 2003.

[9] D. Thorsley and D. Teneketzis, "Diagnosability of stochastic discrete-event systems," *IEEE Trans. Autom. Control*, vol. 50, no. 4, pp. 476–492, Apr. 2005.

[10] O. Contant, S. Lafortune, and D. Teneketzis, "Diagnosability of discrete event systems with modular structure," *Discrete Event Dyn. Syst.-Theory Appl.*, vol. 16, no. 1, pp. 9–37, 2006.

[11] R. Kumar and S. Takai, "Inference-based ambiguity management in decentralized decision-making: Decentralized diagnosis of discrete-event systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 3, pp. 479–491, 2009.

[12] E. Athanasopoulou, L. Lingxi, and C. Hadjicostis, "Maximum likelihood failure diagnosis in finite state machines under unreliable observations," *IEEE Trans. Autom. Control*, vol. 55, no. 3, pp. 579–593, Mar. 2010.

[13] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial algorithm for testing diagnosability of discrete-event systems," *IEEE Trans. Autom. Control*, vol. 46, no. 8, pp. 1318–1321, Aug. 2001.

[14] T.-S. Yoo and S. Lafortune, "Polynomial-time verification of diagnosability of partially observed discrete-event systems," *IEEE Trans. Autom. Control*, vol. 47, no. 9, pp. 1491–1495, Sep. 2002.

[15] W. Qiu and R. Kumar, "Decentralized failure diagnosis of discrete event systems," *IEEE Trans. Syst., Man, Cybern. A*, vol. 36, no. 2, pp. 384–395, Mar. 2006.

[16] Y. Wang, T.-S. Yoo, and S. Lafortune, "Diagnosis of discrete event systems using decentralized architectures," *Discrete Event Dyn Syst*, vol. 17, pp. 233–263, 2007.

[17] J. C. Basilio and S. Lafortune, "Robust codiagnosability of discrete event systems," in *Proc Amer. Control Conf.*, St. Louis, MO, 2009, pp. 2202–2209.

[18] R. Debouk, S. Lafortune, and D. Teneketzis, "Coordinated decentralized protocols for failure diagnosis of discrete event systems," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 10, no. 1, pp. 33–86, 2000.

[19] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event System*. Secaucus, NJ: Springer-Verlag, 2008.

[20] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Boston, MA: Addison Wesley, 2006.

[21] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE, Special Issue Discrete Event Syst.*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2007.

Stability Analysis of Positive Switched Linear Systems With Delays

Xingwen Liu and Chuangyin Dang

***Abstract*—This technical note addresses the stability problem of delayed positive switched linear systems whose subsystems are all positive. Both discrete-time systems and continuous-time systems are studied. In our analysis, the delays in systems can be unbounded. Under certain conditions, several stability results are established by constructing a sequence of functions that are positive, monotonically decreasing, and convergent to zero as time tends to infinity (additionally continuous for continuous-time systems). It turns out that these functions can serve as an upper bound of the systems' trajectories starting from a particular region. Finally, a numerical example is presented to illustrate the obtained results.**

***Index Terms*—Delays, dual systems, positive switched linear system (PSLS), stability.**

I. INTRODUCTION

A switched system is a type of hybrid dynamical system that combines discrete states and continuous states. Informally, it consists of a family of dynamical subsystems and a rule, called a switching signal, that determines the switching manner between the subsystems. Many dynamical systems can be modeled as such switched systems [1]–[3]. Switched systems possess rich dynamics due to the multiple subsystems and various possible switching signals [4], [5]. Many interesting and challenging issues in switched systems have attracted a lot of attention [6]–[9].

Recently, the importance of positive switched linear systems (PSLSs) has been highlighted by many researchers due to their broad applications in communication systems [10], formation flying [11], and systems theory [12], [13]. A positive system implies that its states and outputs are nonnegative whenever the initial conditions and inputs are nonnegative [14], [15]. A PSLS means a switched linear system in which each subsystem is itself a positive system. Positive systems have numerous applications in areas such as economics, biology, sociology and communications [16], [17], [30]. It is well known that positive systems have many special and interesting properties. For example, their stability is not affected by delays [18]–[20]. It should be pointed out that studying the dynamics of positive switched systems is more challenging than that of general switched system because, in order to obtain some "elegant" results, one has to combine the features of positive systems and switched systems [21].

A mass of literature is concerned with the issue of stability of PSLSs [22]–[25]. When stability of positive systems is considered, it is natural

Manuscript received April 05, 2010; revised September 01, 2010, November 20, 2010, and February 10, 2011; accepted February 11, 2011. Date of publication March 03, 2011; date of current version July 07, 2011. This work was supported in part by GRF: CityU 113308 of the Government of Hong Kong SAR, National Nature Science Foundation under Grant 60974148, the Program for New Century Excellent Talents in University under Grant NCET-10-0097, and the Sichuan Youth Science and Technology Fund under Grant 2011JQ0011. Recommended by Associate Editor J. H. Braslavsky.

X. Liu is with the College of Electrical and Information Engineering, Southwest University for Nationalities of China, Chengdu, Sichuan, 610041, China (e-mail: xingwenliu@gmail.com).

C. Dang is with the Department of Manufacturing Engineering and Engineering Management, City University of Hong Kong, Kowloon, Hong Kong, China (e-mail: mecdang@cityu.edu.hk).

Digital Object Identifier 10.1109/TAC.2011.2122710