Discrete Event Dyn Syst (2017) 27:701–739
DOI 10.1007/s10626-017-0260-y

Empirical studies in the size of diagnosers and verifiers for diagnosability analysis

Leonardo Bermeo Clavijo[1] · **João C. Basilio**[2]

Received: 25 August 2016 / Accepted: 25 July 2017 / Published online: 10 August 2017

Abstract Diagnosability is an intrinsic property of the language generated by discrete event systems (DES) and the computational procedure to determine whether a language possesses or not this property is called diagnosability verification. For regular languages, diagnosability verification is carried out by building either diagnoser or verifier automata; the former is known to have worst-case exponential complexity whereas the latter has polynomial complexity in the size of state space of the automaton that generates the language. A question that has been asked for some time now is whether, in average, the state size of diagnosers is no longer exponential. This claim has been supported by the size of diagnoser automata usually obtained in practical and classroom examples, having, in some cases, state space size much smaller than that of verifiers. In an effort to clarify this matter, in this paper we carry out an experimental study on the average state size of diagnosers and verifiers by means of two experiments: *(i)* an *exhaustive experiment*, in which ten sets of automata with moderate cardinality were generated and for these sets of automata, diagnosers and verifiers were built, being the exact average state size for these specific instances calculated; *(ii) an experiment with sampling*, which considers 1660 sets of different instance sizes and, for each one, sample sets of 10,000 automata are randomly generated with uniform distribution and we compute sets of diagnosers and verifiers for each set of randomly generated

This work was carried out while L. B. Clavijo was a D.Sc. student at the Electrical Engineering Post-graduation Program of the Federal University of Rio de Janeiro.

✉ João C. Basilio
basilio@dee.ufrj.br

Leonardo Bermeo Clavijo
lbermeoc@unal.edu.co

[1] Departamento de Ingeniería Eléctrica y Electrónica, Universidad Nacional de Colombia, Cra. 30 #45-03, Edif. 453, Of. 202, Bogotá D.C., Colombia

[2] Departamento de Engenharia Elétrica, Universidade Federal do Rio de Janeiro, 21949-900, Rio de Janeiro, RJ, Brazil

automata, which have been used to estimate an asymptotic model for the average state sizes of diagnosers and verifiers.

Keywords Discrete-event systems · Automaton · Language diagnosability verification · Scientific computation

1 Introduction

Diagnosability is an intrinsic property of the language generated by DES and the computational procedure to determine whether a DES possesses or not this property is called diagnosability verification. For regular languages, diagnosability verification is carried out by building either diagnoser or verifier automata. The use of diagnosers in diagnosability verification has been proposed by Sampath et al. (1995), and has been widely used since then. Diagnosers are known to have worst-case exponential state space complexity in the number of states of the automaton that generates the language, i.e, $O(2^{2n})$. On the other hand, there are several diagnosability verification methods using verifiers (Jiang et al. 2001; Yoo and Lafortune 2002; Qiu and Kumar 2006; Moreira et al. 2011), which can have computational complexity as low as quadratic (Yoo and Lafortune 2002; Qiu and Kumar 2006; Moreira et al. 2011) in the number of the states of the automaton that generates the language, i.e, $O(n^2)$).

A question that has been asked for some time now is whether, in average, state size of diagnosers is no longer exponential; this claim has been supported by the size of diagnoser automata usually obtained in practical and classroom examples, having, in some cases, state space size much smaller than that of the verifiers calculated in accordance with Jiang et al. (2001), Yoo and Lafortune (2002), Qiu and Kumar (2006), and Moreira et al. (2011). In this paper, we intend to partially answer this question by carrying out empirical experiments to estimate the average state size of these devices. We will study the average state size of the diagnoser proposed by Sampath et al. (1995). In the case of verifiers, there exist several algorithms reported in the literature for their construction: the algorithm presented by Jiang et al. (2001) has a worst-case computational complexity of fourth order in the number of states of the system to be diagnosed (i.e., of order $\mathcal{O}(n^4)$), whereas the algorithms reported by Moreira et al. (2011), Qiu and Kumar (2006), and Yoo and Lafortune (2002), all share the same worst-case computational complexity of quadratic order (i.e., of order $\mathcal{O}(n^2)$). In this experimental study, we will select the algorithm proposed by Moreira et al. (2011), for the following reasons (Moreira et al. 2016): *(i)* it is deterministic and is defined using only basic operations over deterministic finite automata; *(ii)* there is some evidence that its state size is usually smaller than other verifiers computed with algorithms that lead to verifiers with the same worst-case computational complexity.

In summary, in this paper we carry out an empirical study on the average state size of the diagnoser proposed by Sampath et al. (1995) and the verifier proposed by Moreira et al. (2011). For this matter, we carry out two experiments: *(i)* an *exhaustive experiment*, in which ten sets of automata with moderate cardinality were generated by using an algorithm proposed here that extends the results by Reis et al. (2005) and Bassino et al. (2009), and for these sets of automata, diagnosers and verifiers were computed, being the exact average state size for these specific instances calculated; *(ii) an experiment with sampling*, which considers 1,660 sets of different instance sizes and, for each one, sample sets of 10,000 automata are randomly generated with uniform distribution according to the method

proposed in Bassino et al. (2009) and Bassino and Nicaud (2007), and then, we compute sets of diagnosers and verifiers for each set of randomly generated automata (33,200,000 diagnosers and verifiers altogether), which have been used to estimate an asymptotic model for the average state size of diagnosers and verifiers. It is worth remarking that the results in enumeration and random generation of automata used in this paper have been previously obtained by the computer science community and, to the best of our knowledge, they have never been deployed by the DES community, despite its practical relevance in the complete evaluation of the real performance of algorithms.

The remainder of this paper is structured as follows. In Section 2, we present a brief review on the construction of diagnosers and verifers. In Section 3, we review the method of exhaustive generation of complete finite deterministic accessible automata and extend this method to exhaustive generation of accessible (but not necessarily complete) automata. We also present the results of an exhaustive experiment for the analysis of the average state size of diagnosers and verifiers. In Section 4, we first review the uniform random generation of accessible automata, and in the sequel, we propose an experiment for determining the average state size of diagnosers and verifiers; based on the experiments, we present empirical models for their average state sizes. Finally, in Section 5, we highlight the main contributions of the paper.

2 Failure diagnosability of DES modelled by automata using diagnosers and verifiers

2.1 Preliminaries

The theoretical foundations of fault diagnosis and diagnosability analysis of DES, as proposed by Sampath et al. (1995), are based on regular languages and automaton theory as formal modeling tools. In that approach, the DES is modeled using a deterministic automaton, which will be denoted here as

$$G = (X, \Sigma, f, \Gamma, X_m, x_0), \tag{1}$$

where X is the finite state space, Σ is the set of events, f is the transition function, assumed to be partially defined over the event set, Γ is the active event set, i.e., $\Gamma(x) = \{\sigma \in \Sigma : (\exists y \in X)[f(x, \sigma) = y]\}$, X_m is the set of marked states, and x_0 is the initial state. In addition, event set Σ is partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o and Σ_{uo} denote, respectively, the sets of observable and unobservable events. The unobservable events are those whose occurrence cannot be detected (including possible fault events) by sensors attached to the system.

The accessible part of G, denoted as $Ac(G)$, is obtained by eliminating all states of G (and their transitions) that are not reachable from the initial state x_0. Formally, $Ac(G) = (X_{ac}, \Sigma, f_{ac}, x_0)$, where $X_{ac} = \{x \in X : (\exists s \in \Sigma^*)[f(x_0, s) = x]\}$ and $f_{ac} : X_{ac} \times \Sigma \to X_{ac}$ is the new transition function obtained by restricting the domain of f to the reduced domain of the accessible states X_{ac}. The coaccessible part of G, denoted as $CoAc(G)$, is obtained by deleting all states of G from which it is not possible to reach a marked state. Formally, $CoAc(G) = (X_{coac}, \Sigma, f_{coac}, X_m, x_{0_{coac}})$, where $X_{coac} = \{x \in X : (\exists s \in \Sigma^*)[f(x, s) \in X_m]\}$, $x_{0_{coac}} = x_0$ 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 when the domain of f is restricted to the set of coaccessible states X_{coac}.

An important binary operation between automata is the parallel composition. Let $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{0_1})$ and $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{0_2})$ denote two automata. The parallel composition between G_1 and G_2, denoted by $G_1 \parallel G_2$, is defined as the automaton:

$$G_1 \parallel G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\|2}, \Gamma_{1\|2}, (x_{0_1}, x_{0_2})),$$

where the transition function $f_{1\|2}$ is defined as follows:

$$f_{1\|2} = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \\ (f_1(x_1, \sigma), x_2), & \text{if } \sigma \in \Gamma_1(x_1) \setminus \Sigma_2, \\ (x_1, f_2(x_2, \sigma)), & \text{if } \sigma \in \Gamma_2(x_1) \setminus \Sigma_1, \\ \text{undefined}, & \text{otherwise.} \end{cases} \tag{2}$$

Let us now assume that a deterministic automaton G has unobservable events. Then, we can represent its observable dynamic behavior through a deterministic automaton called observer which will be denoted here as

$$Obs(G) = (X_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, x_{0_{obs}}). \tag{3}$$

Every state $x_{obs} \in X_{obs}$ is a set containing all the possible states of automaton G where it can be after a recorded sequence of observable events and, therefore, $X_{obs} \in 2^X$, where 2^X denotes the power set of X. In order to construct $Obs(G)$, we need to define the *unobservable reach* of each state $x \in X$, denoted by $UR(x)$, as follows:

$$UR(x) = \{y \in X : (\exists t \in \Sigma_{uo}^*)[f(x, t) = y]\}. \tag{4}$$

This definition is extended to sets of states $Q \subseteq X$, as follows:

$$UR(Q) = \cup_{x \in Q} UR(x). \tag{5}$$

Algorithm 1 presents the procedure for building observer automata.

Algorithm 1 Construction of an observer automaton for automaton G

Inputs: Automaton $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ and its set of observable events Σ_o.
Output: *Deterministic automaton* $G_{obs} = Obs(G) = (X_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, x_{0,obs})$.

STEP 1: Compute $x_{0_{obs}} = UR(x_0)$
Set $X_{obs}^{new} = \{x_{0_{obs}}\}$ and $X_{obs} = \emptyset$

STEP 2: Set $X_{obs} = X_{obs} \cup X_{obs}^{new}$, $X_{obs}^{cur} = X_{obs}^{new}$, and $X_{obs}^{new} = \emptyset$

STEP 3: For each $Q \in X_{obs}^{cur}$ and each $\sigma \in \Sigma_o$, compute
$Q' = UR(\{x \in X : (\exists x_\sigma \in Q)[x \in f(x_\sigma, \sigma)]\})$.

STEP 4: If $Q' \neq \emptyset$
Set $f_{obs}(Q, \sigma) = Q'$
Set $X_{obs}^{new} = X_{obs}^{new} \cup \{Q'\}$

STEP 5: Repeat steps 2 and 3 until the entire accessible part of $Obs(G)$ has been generated.

Remark 1 Denoting by n the state space cardinality of automaton G, then, the worst case computational complexity in the construction of automaton $Obs(G)$ is 2^n (i.e., of order $\mathcal{O}(2^n)$) (see Moore (1971); Wong (1998)).

2.2 Fault diagnosis of DES

Let us denote by L the language generated by automaton G. Let $\Sigma_f = \{\sigma_f\} \subseteq \Sigma_{uo}$ denote the singleton containing the failure event σ_f whose occurrence must be *diagnosed*, i.e., we want to infer without ambiguity the occurrence of σ_f, within a finite delay, by observing only events in Σ_o. This is indeed a property of language L called diagnosability.

The following assumptions are usually made (Sampath et al. 1995):

A1. Language L is live, i.e., $\Gamma(x_i) \neq \emptyset$ for all $x_i \in X$.
A2. There is no cycles of unobservable events in G.

Notice that, there is no loss of generality in assuming that $\Sigma_f = \{\sigma_f\}$, since if it is assumed that there exist more than one failure event, the approach considered here is still valid. In this regard, let $\Sigma_f = \{\sigma_{f,1}, \sigma_{f,2}, \ldots, \sigma_{f,n}\}$, where $\sigma_{f,i}$, $i = 1, 2, \ldots, n$ are distinct failure events. Then, language diagnosability can be analyzed separately for each event $\sigma_{f,i}$, $i = 1, 2, \ldots, n$, by considering events $\sigma_{f,j}$, $j \neq i$, as ordinary unobservable events (see Yoo and Lafortune (2002)).

Let us assume that $\Psi(\Sigma_f)$ denote the set of all traces of L that ends with failure event σ_f. By a somewhat minor abuse of notation, we use $\Sigma_f \in s$ to denote that $\overline{s} \cap \Psi(\Sigma_f) \neq \emptyset$, where $\overline{s} = \{u \in \Sigma^* : (\exists v \in \Sigma^*)[uv = s]\}$. Therefore, $s \in L$ is a trace containing the failure event σ_f if $\Sigma_f \in s$. We also denote $L/s = \{t \in \Sigma : st \in L\}$ as the language continuation of L after s. Thus, language diagnosability can be formally stated as follows.

Definition 1 (Sampath et al. 1995) A live and prefix-closed language L is diagnosable with respect to $P_o : \Sigma^* \to \Sigma_o^*$ and Σ_f if

$$(\exists n \in \mathbb{N})(\forall s \in \Psi(\Sigma_f))(\forall t \in L/s, |t| \geq n) \Rightarrow (\forall \omega \in P_o^{-1}[P_o(st)] \cap L)[\Sigma_f \in \omega], \quad (6)$$

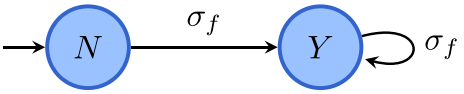
where P_o denotes the usual language projection operation (Ramadge and Wonham 1989) which is defined as follows: *(i)* $P_o(\sigma) = \sigma$, if $\sigma \in \Sigma_o$; *(ii)* $P_o(\sigma) = \varepsilon$, if $\sigma \notin \Sigma_o$; *(iii)* $P_o(s\sigma) = P_o(s)P_o(\sigma)$, for $s \in \Sigma^*$ and $\sigma \in \Sigma$, with ε denoting the empty trace.

The first test for diagnosability verification of regular language was proposed by Sampath et al. (1996) and uses the so-called diagnoser automaton which is obtained as follows:

$$G_D = (X_d, \Sigma_o, f_d, \Gamma_d, x_{0_d}) = Obs(G \parallel A_l) \quad (7)$$

where $A_l = (X_l, \Sigma_f, f_l, x_{o,l})$ is the two state label automaton depicted in Fig. 1. A state $x_d \in X_d$ is called certain (or faulty), if $\ell = Y$ for all $(x, \ell) \in x_d$, and normal (or non-faulty) if $\ell = N$ for all $(x, \ell) \in x_d$. If there exist $(x, \ell), (y, \tilde{\ell}) \in x_d$, x not necessarily distinct from y such that $\ell = Y$ and $\tilde{\ell} = N$, then x_d is an uncertain state of G_d. When the diagnoser is in a certain (normal) state, it is certain that a fault has (resp. has not) occurred. However, if the diagnoser is in an uncertain state, it is not sure if the fault event has occurred or not. As a consequence, if there exists a cycle formed with uncertain states, only, where

Fig. 1 Label automaton A_l



the diagnoser can remain forever, then it will never be able to diagnose the fault occurrence; on the other hand if somehow it always leaves this cycle of uncertain states, then this cycle is not indeterminate. The following definition presents the conditions a cycle of uncertain states must satisfy in order to be an indeterminate cycle.

Definition 2 (Indeterminate cycles of G_d) A set of uncertain states $\{x_{d_1}, x_{d_2}, \ldots, x_{d_p}\} \subset X_d$ forms an indeterminate cycle if the following conditions hold true:

(i) $x_{d_1}, x_{d_2}, \ldots, x_{d_p}$ form a cycle in G_d;

(ii) $\exists (x_l^{k_l}, Y), (\tilde{x}_l^{r_l}, N) \in x_{d_l}$, $x_l^{k_l}$ not necessarily distinct from $\tilde{x}_l^{r_l}$, $l = 1, 2, \ldots, p$, $k_l = 1, 2, \ldots, m_l$, and $r_l = 1, 2, \ldots, \tilde{m}_l$ in such a way that the sequence of states $\{x_l^{k_l}\}$, $l = 1, 2, \ldots, p$, $k_l = 1, 2, \ldots, m_l$ and $\{\tilde{x}_l^{r_l}\}$, $l = 1, 2, \ldots, p$, $r_l = 1, 2, \ldots, \tilde{m}_l$ form cycles in G;

(iii) there exist $s = s_1 s_2 \ldots s_p \in \Sigma^*$ and $\tilde{s} = \tilde{s}_1 \tilde{s}_2 \ldots \tilde{s}_p \in \Sigma^*$ such that $P_o(s) = P_o(\tilde{s}) \neq \epsilon$, where $s_l = \sigma_{l,1}\sigma_{l,2}\ldots\sigma_{l,m_l-1}$, $f(x_l^j, \sigma_{l,j}) = x_l^{j+1}$, $j = 1, 2, \ldots, m_l - 1$, $f(x_l^{m_l}, \sigma_{l+1,0}) = x_{l+1}^1$, and $f(x_p^{m_p}, \sigma_{1,0}) = x_1^1$, and similarly for $\tilde{s}_l$.

Using Definition 2, a necessary and sufficient condition for language diagnosability based on the diagnoser automaton G_D, constructed in accordance with Eq. 7, was presented in Sampath et al. (1995), as follows.

Theorem 1 *The language L generated by automaton G is diagnosable with respect to projection P_o and $\Sigma_f = \{\sigma_f\}$ if, and only if, its diagnoser G_d has no indeterminate cycles.*

Notice that the worst-case computational complexity in computing G_D is 2^{2n} (i.e., of order $\mathcal{O}(2^{2n})$) due to the computation of the observer (7) for automaton $G \parallel A_l$, which has $2n$ states in the worst case.

Diagnosability verification of regular languages can also be carried out in polynomial time by using the verifier automaton proposed by Moreira et al. (2011), which has worst-case computational complexity of $\mathcal{O}(n^2)$ and is built in accordance with Algorithm 2. Notice that the verifier state x_V has two components, i.e., $x_V = (x_N, x_Y)$, where x_N (resp. x_Y) comes from automaton G_N (resp. G_Y), and is always equal to xN (resp. either equal to xN or xY), with N and Y being labels that indicate that the failure event has occurred or not. Thus, based on the verifier constructed in accordance with Algorithm 2, the following necessary and sufficient condition for language diagnosability can be stated.

Theorem 2 *The language L, generated by G, is not diagnosable with respect to P_o and $\Sigma_f = \{\sigma_f\}$ if and only if there exists a cycle $cl := (x_V^k, \sigma_k, x_V^{k+1}, \ldots, x_V^l, \sigma_l, x_V^k)$, where $l \geq k \geq 0$, in G_V satisfying the following conditions:*

$$\exists j \in \{k, k+1, \ldots, l\},\ \textit{s.t. for some } x_V^j = (x_N^j, x_Y^j) \Rightarrow (x_Y^j = xY) \wedge (\sigma_j \in \Sigma).$$

Algorithm 2 Construction of verifier automaton (Moreira et al. 2011)

Inputs: Automaton G whose diagnosability must be verified and its set of unobservable events Σ_{uo}.

Output: Verifier automaton G_V.

STEP 1: Starting from automaton G, compute automaton G_N that models the normal behavior of G, as follows::

- *Step 1.1:* Obtain automaton G_{N1} by deleting the event σ_f and the transitions in G labeled with this event.
- *Step 1.2:* Compute $G_N = Ac(G_{N1}) = (X_N, \Sigma_N, f_N, \Gamma_N, x_{0N})$, where $\Sigma_N = \Sigma \setminus \Sigma_f$.

STEP 2: Compute automaton G_Y that models the failure behavior of the system, as follows:

- *Step 2.1:* Compute $G_l = G \parallel A_l$, where A_l is the label automaton of Fig. 1, and mark all states whose second component is Y.
- *Step 2.2:* Compute $G_Y = CoAc(G_l)$.

STEP 3: Compute automaton $G_{N,R}$ by renaming all unobservable events of automaton G_N according to mapping $R : \Sigma_n \to \Sigma_R$ which is defined by

$$R = \{(\sigma, \sigma_R) : \sigma \in \Sigma_{uo}\}.$$

STEP 4: Compute the verifier automaton $G_V = G_{N,R} \parallel G_Y = (X_V, \Sigma_R \cup \Sigma, f_V, \Gamma_V, x_{0,V})$.

3 Exhaustive experimental analysis on the state sizes of diagnosers and verifiers

In this section we will present preliminary results on the average state sizes of diagnosers and verifiers. To this end, we will analyze automaton classes for which it is feasible (as far as the number of all different automata in the class) to generate the whole class. Therefore, we will start by presenting a systematic way of generating all automata with prespecified number of states and events.

3.1 Exhaustive generation of complete finite deterministic accessible automata

We start by reviewing the method of exhaustive generation of automata proposed by Reis et al. (2005). This method has, as its main feature, the possibility of representing an automaton with a string of digits, therefore using a minimal amount of memory to represent deterministic automata.

Definition 3 (Isomorphism between automata) Let $G = (X, \Sigma, f, \Gamma, x_0)$ and $G' = (X', \Sigma, f', \Gamma', x_0')$ be two automata defined over the same input alphabet Σ. An isomorphism φ from G to G', denoted by $\varphi : G \to G'$, is a mapping $\varphi : X \to X'$ satisfying the following three conditions:

i. φ is a bijection;
ii. $\varphi(x_0) = x_0'$;

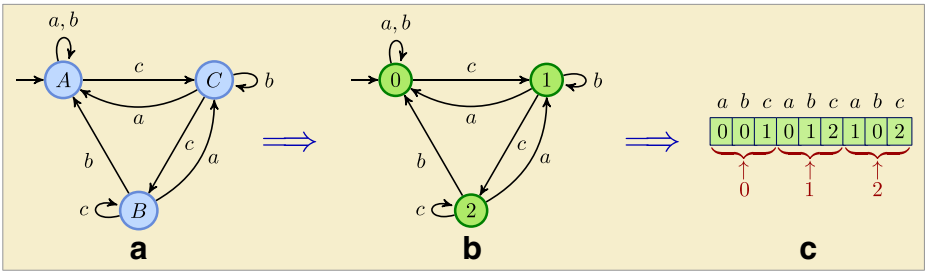


Fig. 2 Automaton G (**a**); Automaton G' isomorphic with automaton G (**b**); Representative sequence of automaton G' (**c**)

iii. $\varphi(f(x,\sigma)) = f'(\varphi(x),\sigma)$.

Let $G = (X, \Sigma, f, \Gamma, x_0)$ denote a complete finite deterministic accessible automaton (CFDAA) such that $|X| = n$ and $|\Sigma| = k$ and assume that alphabet Σ possesses a total order relation $\preccurlyeq$.[1]

Therefore, the depth–first search traversal of automaton G induces an order over the set of states X if, at each visited state, we select the transition labeled with the minimal event according to the order of alphabet Σ. In this regard, we define an isomorphism $\varphi : G \to G'$ ($G' = (X', \Sigma, f', \Gamma', 0)$) where $X' = \{0, 1, \ldots, n-1\}$. For instance, consider the automaton depicted Fig. 2a where the set $\Sigma = \{a, b, c\}$ is ordered alphabetically. The depth–first search induces the bijection $\varphi = \{(A, 0), (C, 1), (B, 2)\}$ between the sets of states X and $X' = \{0, 1, 2\}$ whose elements represent the order in which the states of automaton G were visited. Note that the initial state is numbered as 0.

Figure 2b shows automaton G' which is isomorphic to automaton G through the bijection φ. As depicted in Fig. 2c, we can represent automaton G' by the word 001012102 defined in base n and composed of nk symbols. Each group of k symbols determines the states that will be reached by automaton G' when the transition labeled by the event that has the same position in Σ, according to order relation $\preccurlyeq$, is triggered. For example, digit 1 in the third position indicates that state 0 is connected to state 1 by event c, whereas the same digit in the fith position indicates that there is a self-loop in state 1 labeled by event b.

Definition 4 (Representative sequence) Let $G = (X, \Sigma, f, \Gamma, x_0)$ be a CFDAA such that $|X| = n$ and let $\Sigma = \{\sigma_1 \prec \cdots \prec \sigma_k\}$ be a completely ordered alphabet. Denote by $I_{m:n} = \{m, m+1, \ldots n\} \subseteq \mathbb{Z}$. Let $\varphi : X \to \{0, 1, \ldots, n-1\}$ be an isomorphism from X to X' induced by performing a depth–first search traversal of automaton G following, at each state, the transition labeled with the smallest event, according to the order of Σ. Let $G' = (X', \Sigma, f', \Gamma', 0)$ be the automaton isomorphic to G that is obtained by using φ. The *representative sequence* of G' has the form[2] $(s_i)_{i \in I_{0:kn-1}} = s_0 s_1 \ldots s_{kn-1}$ with $s_i \in I_{0:n-1}$, and $s_i = f'(\lfloor i/k \rfloor, \sigma_{(i \mod k)+1})$, where $k = |\Sigma|$ and $i \in I_{0:kn-1}$.

[1] A set A possesses a total order relation, denoted as $\preccurlyeq$, if for any $a, b, c \in A$ the following conditions are satisfied: *(i)* $a \preccurlyeq a$ for any a (reflexivity); *(ii)* if $a \preccurlyeq b$ and $b \preccurlyeq a$, then $a = b$ (antisymmetry); *(iii)* if $a \preccurlyeq b$ and $b \preccurlyeq c$, then $a \preccurlyeq c$ (transitivity); *(iv)* either $a \preccurlyeq b$ or $b \preccurlyeq a$ (totality).

[2] The notation $\lfloor x \rfloor$ denotes the *integer part of* x, which is defined as $\lfloor x \rfloor = \max\{m \in \mathbb{N} : m \leq x\}$.

Theorem 3 (Reis et al. 2005) *Let* (s_i) *be a representative sequence satisfying the following conditions:*

R1. $(\forall m \in I_{2:n-1})(\forall i \in I_{0:kn-1})(s_i = m \Rightarrow (\exists j \in I_{0:i-1})[s_j = m-1])$,
R2. $(\forall m \in I_{1:n-1})(\exists j \in I_{0:km-1})[s_j = m]$.

Then there exists a bijection between the set of sequences that satisfy **R1** *and* **R2** *and the set of CFDAA* $\mathcal{C}^{(n,k)}$.

Rule **R1** restricts a state $x' \in X'$, labeled by a number greater than zero, to appear in the sequence (s_i) only after the occurrence of its predecessors. This is a direct consequence of the order of the states induced by isomorphism φ. For example, in the sequence 001012102 of Fig. 2c, $s_5 = 2$ ($m = 2$ and $i = 5$ in rule **R1**), which implies that at least one of the symbols in the subsequence $s_0 \cdots s_4 = 00101$ must be 1. Rule **R2** establishes a condition for the accessibility of the automaton whose representative sequence is (s_i). In words, **R2** implies that a state labeled by m must appear at least once in the first $km-1$ positions of the representative sequence. For instance, rule **R2** applied to the sequence of Fig. 2c with $m = 1$ and $k = 3$, determines that at least one of the symbols in the subsequence $s_0 s_1 s_2 = 001$ must be 1.

In order to generate all CDFAA of $\mathcal{C}^{(n,k)}$, rules **R1** and **R2** must be reformulated in a more suitable form using *flags*. Let (f_j), $j \in I_{1:n-1}$, $f_j \in I_{0:kn-1}$ be a sequence of flags where the symbol f_j represents the position of the first occurrence of the state labelled by j in the representative sequence $(s_i)_{i \in I_{0:kn-1}}$. For instance, in the sequence 001012102 of Fig. 2c, $f_j \in I_{0:8}$, $j \in I_{1:2}$, $f_1 = 2$ and $f_2 = 5$.

In terms of the sequence of flags $(f_j)_{j \in I_{1:n-1}}$, **R1** and **R2** correspond, respectively, to the following rules:

G1. $f_{j-1} < f_j, \forall j \in I_{2:n-1}$,
G2. $f_m \leq km-1, \forall m \in I_{1:n-1}$.

Rules **G1** and **G2** imply that $f_1 \in I_{0:k-1}$ and $f_{j-1} < f_j < kj$ for $j \in I_{2:n}$. Notice that if we consider sequence 001012102, then rule **G1** with $j = 2$ determines the existence of a flag $f_1 < f_2 = 5$, which in turn is equivalent to state that if $s_5 = 2$, then at least one of the symbols in the subsequence $s_0 \cdots s_4$ must be equal to 1 (rule **R1**). To use the same example as above, rule **G2** with $m = 1$ establishes that $f_1 \in I_{0:2}$ which implies that at least one of the symbols of the subsequence $s_0 s_1 s_2$ must be equal to 1 (rule **R2**).

In order to generate all CDFAA corresponding to a fixed sequence of flags $(f_j)_{j \in I_{0:n}}$, the symbols s_i, where $i \notin \{f_j : j \in I_{1:n-1}\} \cup I_{0:f_1}$ (i.e., all symbols that were not defined by the sequence of flags $(f_j)_{j \in I_{1:n}}$), must be set according to the following rules:

G3. $s_i = 0$, for $i < f_1$;
G4. $\forall j \in I_{1:n-2}, s_i \in I_{0:j}$, for $f_j < i < f_{j+1}$;
G5. $s_i \in I_{0:n-1}$, for $i > f_{n-1}$.

Rules **G3–G5** are constructive rules for obtaining a well-formed sequence, i.e., a sequence representing an automaton, starting from a given sequence of flags. Taking again sequence 001012102 as an example, we can see that rule **G3** constrains subsequence $s_0 s_1 = 00$, to the left of $s_{f_1} = s_2$ to be formed with zeros. Rule **G4** restricts the symbols of subsequence $s_3 s_4 = 01$, that appear between the symbols $s_{f_1} = s_2$ and $s_{f_2} = s_5$, to be digits in base 2.

The theory outlined above provides all of necessary technical details for introducing Algorithm 3, which exhaustively generates every finite deterministic accessible automaton of the set of complete automata $\mathcal{C}^{(n,k)}$. As it can be noted, Algorithm 3 is composed of two main procedures: *recursive generation of sequences* and *recursive generation of flags*, as follows:[3]

Algorithm 3 Exhaustive generation of set $C^{(n,k)}$

Input: $n = |X|$ *and* $k = |\Sigma|$, *the cardinalities of the sets of states and events of automata* $G \in C^{(n,k)}$.

Output: *The set* $C^{(n,k)}$ *of complete automata.*

$(f_j)_{j\in I_{1:n-1}} \longleftarrow (kj-1)_{j\in I_{1:n-1}}$ *(initial sequence of flags)*

$(s_i)_{i\in I_{0:kn-1}} \longleftarrow 0^{k-1}10^{k-1}\dots(n-2)0^{k-1}(n-1)0^k$ *(initial representative sequence)*

$C^{(n,k)} \longleftarrow$ {automaton associated to initial representative sequence $(s_i)_{i\in I_{0:kn-1}}$}

$s_f \longleftarrow 12\dots(n-1)(n-1)^{(k-1)n+1}$ *(final representative sequence)*

while $(s_i)_{i\in I_{0:kn-1}} \neq s_f$ **do**

  $(s_i)_{i\in I_{0:kn-1}} \longleftarrow$ GENERATE SEQUENCE$((s_i)_{i\in I_{0:kn-1}}, (f_j)_{j\in I_{1:n-1}})$

  if $(s_i)_{i\in I_{0:kn-1}} = 0^{nk}$ **then**

    $(f_j)_{j\in I_{1:n-1}} \longleftarrow$ GENERATEFLAGS$((f_j)_{j\in I_{1:n-1}})$

    for $j \in I_{1:n-1}$ **do**

      $s_{f_j} = j$ *(initial sequence of flags)*

  $C^{(n,k)} \longleftarrow C^{(n,k)} \cup$ {automaton associated to sequence $(s_i)_{i\in I_{0:kn-1}}$}

1. Recursive generation of sequences (Algorithm 4) Let $(f_j)_{j\in I_{1:n-1}}$ be the sequence of flags of a given representative sequence $(s_i)_{i\in I_{0:kn-1}}$. Starting from representative sequence $(s_i)_{i\in I_{0:kn-1}}$, we can generate a new one by modifying, in a consistent manner, the symbols of $(s_i)_{i\in I_{0:kn-1}}$ (according to rules **G3**–**G5**) that are not fixed by the flags. In the following, we describe how to obtain a new representative sequence. Define set $NI = I_{f_1+1:kn-1} \setminus \{f_j : j \in I_{1:n-1}\}$ of subindexes of the symbols of $(s_i)_{i\in I_{0:kn-1}}$ that were not set by the sequence of flags. According to rules **G4** and **G5**, each symbol s_i such that $i \in NI$ is a number expressed in base $j+1$, where j is the subindex of the first right-most symbol s_j set by flag f_j in the representative sequence $(s_i)_{i\in I_{0:kn-1}}$. Specifically, to determine the base of digit s_i, compute the set

$$B_i = \{i - f_j : (f_j \in \{f_1, \dots, f_{n-1}\}) \wedge (i - f_j > 0)\}$$

and represent by b_i the unique value of l such that $f_l = i - \min(B_i)$, with $f_l \in \{f_1, \dots, f_{n-1}\}$. The following rule establishes how to modify a symbol in the subsequence $(s_i)_{i\in NI}$ in order to obtain a new representative sequence:

$$s_i' = \begin{cases} s_i + 1, & \text{if } s_i + 1 \leq b_i \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

The procedure begins by applying rule (8) to symbol s_{kn-1}. If the condition $s_{kn-1} + 1 \leq b_{kn-1} = f_{n-1}$ is satisfied, then the next representative sequence will be $s_0 \cdots s_{nk-2}(s_{kn-1}+1)$. Otherwise, rule (8) must be propagated to the other symbols of $(s_i)_{i\in NI}$ in descending order of NI until the condition $s_i + 1 \leq b_i$ of rule (8) is satisfied. Notice that if there exists a symbol s_i satisfying $s_i + 1 \leq b_i$, the algorithm returns a new representative sequence

[3] The reader may find useful to follow the explanation with the help of Example 1.

$(s'_i)_{i\in I_{0:kn-1}}$ (see line *l*1), Otherwise, it returns the sequence 0^{nk} (see line *l*2) which does not correspond to any CFDAA, therefore indicating that a complete series of representative sequences, corresponding to a given sequence of flags, has been generated.

Algorithm 4 Function for recursive generation of representative sequences

function GENERATESEQUENCE($(s_i)_{i\in I_{0:nk-1}}$, $(f_j)_{j\in I_{1:n-1}}$)
Input: *a sequence* $(f_j)_{j\in I_{1:n-1}}$ *of flags and a representative sequence* $(s_i)_{i\in I_{0:nk-1}}$
Output: *the next representative sequence* $(s'_i)_{i\in I_{0:nk-1}}$
$NI \longleftarrow I_{f_1+1:kn-1} - \{f_j : j \in I_{1:n-1}\}$
$(s'_i)_{i\in I_{0:nk-1}} \longleftarrow (s_i)_{i\in I_{0:nk-1}}$
for $i \in NI$, *traversing in descending order of* NI **do**
  $B_i \longleftarrow \{i - f_j : (f_j \in \{f_1, \ldots, f_{n-1}\}) \wedge (i - f_j > 0)\}$
  $b_i \longleftarrow$ the value of l such that $f_l = i - \min B_i$, with $f_l \in \{f_1, \ldots, f_{n-1}\}$
  if $s_i + 1 \leq b_i$ **then**
    $s'_i \longleftarrow s_i + 1$
*l*1 **return** $(s'_i)_{i\in I_{0:nk-1}}$
  else
    $s'_i \longleftarrow 0$
*l*2 **return** 0^{nk}

2. Recursive generation of flags (Algorithm 5) Let $(f_j)_{j\in I_{1:n-1}}$ be a given sequence of flags. It is possible to generate a new sequence of flags, $(f'_j)_{j\in I_{1:n-1}}$, by modifying the current sequence of flags $(f_j)_{j\in I_{1:n-1}}$ according to rules **G1**–**G2**, as follows:

$$f'_j = \begin{cases} f_j - 1, & \text{if } j > 1 \text{ and } f_j - 1 > f_{j-1} \\ f_j - 1, & \text{if } j = 1 \\ kj - 1, & \text{otherwise.} \end{cases} \tag{9}$$

Algorithm 5 begins by using rule (9) in symbol f_{n-1}. If the condition $f_{n-1} - 1 > f_{n-2}$ is satisfied, the next sequence of flags will be $f_1 f_2 \cdots f_{n-2}(f_{n-1} - 1)$. Otherwise, rule (9) must be propagated in descending order of $I_{1:n-1}$, for each f_i in sequence $(f_i)_{i\in I_{1:n-2}}$ until either condition $f_j - 1 > f_{j-1}$ is verified, or when symbol f_1 is reached. Algorithm 5 is the pseudo-code implementation of the procedure for recursive generation of flags.

Algorithm 5 Function for recursive generation of flags

function GENERATEFLAGS($(f_j)_{j\in I_{1:n\ \ 1}}$)
Input: *the sequence* $(f_j)_{j\in I_{1:n-1}}$ *of flags.*
Output: *the next generated sequence* $(f_j)_{j\in I_{1:n-1}}$ *of flags* $(f'_j)_{j\in I_{1:n-1}} \longleftarrow (f_j)_{j\in I_{1:n-1}}$
for $j \in I_{1:n-1}$, *traversing in descending order of* $I_{1:n-1}$ **do**
*c*1 **if** $j = 1$ **then**
    $f'_1 \longleftarrow f_1 - 1$
    return $(f'_j)_{j\in I_{1:n-1}}$
*c*2 **if** $(f_j - 1 > f_{j-1})$ **then**
    $f'_j \longleftarrow f_j - 1$
    return $(f'_j)_{j\in I_{1:n-1}}$
  else
    $f'_j \longleftarrow kj - 1$

Table 1 Series of representative of all automata in $C^{(3,2)}$ generated by using Algorithm 3

$f_1 f_2 = 13$	$f_1 f_2 = 12$	$f_1 f_2 = 03$	$f_1 f_2 = 02$	$f_1 f_2 = 01$
(010200,1)	(012000,19)	(100200,46)	(102000,82)	(120000,136)
(010201,2)	(012001,20)	(100201,47)	(102001,83)	(120001,137)
⋮	⋮	⋮	⋮	⋮
(010222,9)	(012022,27)	(100222,54)	(102222,108)	(120222,162)
(011200,10)	(012100,28)	(101200,55)	(112000,109)	(121000,163)
⋮	⋮	⋮	⋮	⋮
(011221,17)	(012221,44)	(111221,80)	(112221,134)	(122221,215)
(011222,18)	(012222,45)	(111222,81)	(112222,135)	(122222,216)

The following example shows how to generate a set $\mathcal{C}^{(n,k)}$ of complete automata in accordance with Algorithm 3.

Example 1 Let us consider the exhaustive generation of set $\mathcal{C}^{(3,2)}$ (i.e., the set of complete automata with $n = 3$ states and $k = 2$ events). Notice that, for this case, each sequence of flags is of the form $(f_j)_{j\in I_{1:n-1}} = f_1 f_2$ and each representative sequence has the form $(s_i)_{i\in I_{0:kn-1}} = s_0 \cdots s_5$. The evolution of Algorithm 3 is summarized in Table 1, whose column are formed with the series of representative sequences associated with a particular sequence of flags, and each element of the column is a representative sequence; the numbers on the right of the comma correspond to the order the sequence was generated.

The algorithm starts with the flag sequence $f_1 f_2 = (kj-1)_{j\in I_{1:n-1}} = 13$ and with the seed representative sequence $0^{k-1}10^{k-1}\ldots(n-2)0^{k-1}(n-1)0^k = 010200$ (element (1,1) of the Table 1). Then, by calling function GENERATESEQUENCE($s_0\cdots s_5$, $f_1 f_2$) (line 6 of Algorithm 3), the series of representative sequences of the first column of the Table 1 is generated. Notice that the last representative sequence of column 1 is 011222, since, after its generation, function GENERATESEQUENCE creates the invalid sequence 000000 that is detected in line 7 of Algorithm 3.

At this stage, function GENERATEFLAGS($f_1 f_2$) (line 8 of Algorithm 3) is used to generate the next sequence of flags, which, in this case, will be $f_1 f_2 = 12$ (label of the second column of Table 1); therefore determining a new seed representative sequence 012000 of Algorithm 3. The process of representative sequence generation is repeated until all columns of Table 1 have been generated.

The following result gives the value for the cardinality of set $\mathcal{C}^{(n,k)}$.

Theorem 4 (Almeida et al. 2007) *The cardinality of set $\mathcal{C}^{(n,k)}$, with $k \geq 1$ and $n > 1$ is given by:*

$$\left|\mathcal{C}^{(n,k)}\right| = \sum_{h_1=0}^{k-1} \sum_{h_2=h_1+1}^{2k-1} \sum_{h_3=h_2+1}^{3k-1} \cdots \sum_{h_{n-1}=h_{n-2}+1}^{k(n-1)-1} \prod_{i=1}^{n} i^{h_i-h_{i-1}-1}, \tag{10}$$

where $h_0 = -1$ and $h_n = kn$.

Notice that, in applying Theorem 4 to set $\mathcal{C}^{(3,2)}$ of CFDAA, then $n = 3$ and $k = 2$. Thus, By using Eq. 10, we obtain the value of $|\mathcal{C}^{(3,2)}|$, as follows:

$$\left|\mathcal{C}^{(3,2)}\right| = \sum_{h_1=0}^{1} \sum_{h_2=h_1+1}^{3} \prod_{i=1}^{3} i^{h_j - h_{j-1}-1}\Big|_{h_0=-1, h_3=6} = 216,$$

which is equal to the number of sequences obtained with the exhaustive generation using Algorithm 3.

3.2 Exhaustive generation of accessible (but not necessarily complete) automata

The method described in the previous subsection allows to generate $\mathcal{C}^{(n,k)}$, the set of complete deterministic and accessible automata, where $|X| = n$ and $|\Sigma| = k$. However, what is really necessary here is to generate sets of accessible, not necessarily complete, automata. Let us denote by $\mathcal{A}^{(n,k)}$ the set of deterministic and accessible automata. In this subsection, we will propose an algorithm for the generation $\mathcal{A}^{(n,k)}$ based on a result that establishes a bijection between accessible automata and a subset of complete automata presented by Bassino et al. (2009). We start with the following definition.

Definition 5 (Representative automaton) Let $G = (X, \Sigma, f, \Gamma, x_0)$ denote an accesible automaton such that $|X| = n$ and $|\Sigma| = k$, where the alphabet $\Sigma = \{\sigma_1 \prec \cdots \prec \sigma_k\}$ is completely ordered. Consider the mapping $\psi : X \to \Sigma^*$, defined for each $x \in X$, as follows:

$$\psi(x) = \min_{\prec_{lex}} \left\{ s \in \Sigma^* : (f(x_0, s) = x) \wedge (s \text{ is a simple path of } G) \right\}, \tag{11}$$

where the minimum is defined in accordance with the lexicographical order $\prec_{lex}$.[4] A representative automaton $G' = (X', \Sigma, f', \Gamma', \epsilon)$ is an automaton isomorphic to G through $\psi' : X \to X'$ where $\psi' : X \to X'$ is the mapping $X' = \{\psi(x) : x \in X\}$, where $\psi(x)$ is defined in Eq. 11.

Notice that, since G is accessible, $\psi(x)$ is defined for all $x \in X$. In addition, since G is a deterministic automaton, $\psi(x_1) \neq \psi(x_2)$ for all $x_1, x_2 \in X, x_1 \neq x_2$, which implies that the mapping $\psi' : X \to X'$ is a bijection; therefore allowing the definition of isomorphism $\psi' : G \to G'$. As a consequence, G', being isomorphic to G, is also accessible and deterministic.

Example 2 Let us consider automaton G whose state transition diagram is shown in Fig. 3a. The application of ψ over each state $x \in X$ leads to:

$$\psi' = \{(x_0, \epsilon), (x_1, aabc), (x_2, aab), (x_3, a), (x_4, aa)\},$$

from which, it is possible to define the representative automaton G' shown in Fig. 3b. The red transitions in automaton G show the minimal simple paths to reach each state of G; for instance, the shortest path to reach state x_4, according to the lexicographical order $\prec_{lex}$, is aa.

[4] The lexicographical order is the order used in a dictionary. Formally, let $\Sigma = \{\sigma_1 \prec \cdots \prec \sigma_k\}$ be an ordered set. The lexicographical order, denoted as $\prec_{lex}$ is the order defined as follows: $s \prec_{lex} t$, if either s is a prefix of t, or $s = p\sigma_1 u'$ and $t = p\sigma_2 t'$ for some $p, u', t' \in \Sigma^*$ and $\sigma_1, \sigma_2 \in \Sigma$ with $\sigma_1 \prec \sigma_2$.

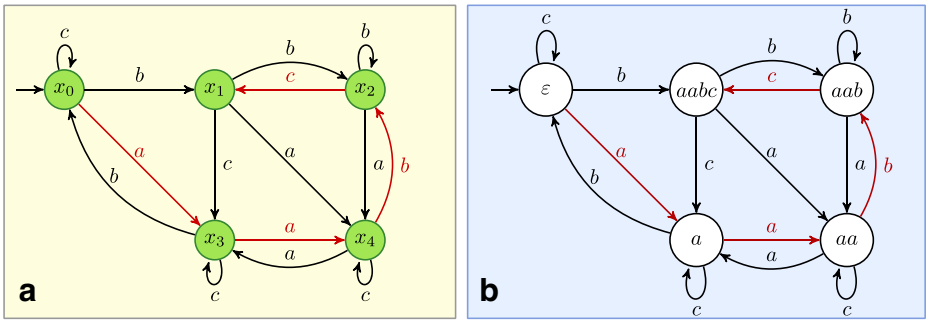


Fig. 3 Automaton $G = (X, \Sigma, f, \Gamma, x_0)$ (**a**); representative automaton $G' = (X', \Sigma, f', \Gamma', \epsilon)$, isomorphic to G

We will now use the bijection relation proposed in Bassino et al. (2009) to propose an algorithm for the generation of accessible (but not necessarily complete) automata. In order to do so, let $A = (X_A, \Sigma, f_A, \Gamma_A, \epsilon) \in \mathcal{A}^{(n,k)}$ denote a representative automaton. Define $\phi : \mathcal{A}^{(n,k)} \to \mathcal{C}^{(n+1,k)}$ as the function that associates automaton A with automaton $\phi(A) = (X_\phi, \Sigma, f_\phi, \Gamma_\phi, \epsilon) \in C^{(n+1,k)}$, where $X_\phi = \{\sigma_k x_a : x_a \in X_A\} \cup \{\epsilon\}$, $\sigma_k = \max(\Sigma)$, and with state transition function f_ϕ defined as follows:

- $$f_\phi(\epsilon, \sigma) = \begin{cases} \epsilon, & \sigma \neq \sigma_k, \\ \sigma_k, & \sigma = \sigma_k. \end{cases} \tag{12a}$$
- $$f_\phi(x_\phi, \sigma) = \begin{cases} \sigma_k f_A(x_A, \sigma), & \text{if } (\exists x_A \in \Sigma^*)[(x_\phi = \sigma_k x_A) \wedge (f_A(x_A, \sigma) \neq \emptyset)], \\ \epsilon, & \text{if } (\exists x_A \in \Sigma^*)[(x_\phi = \sigma_k x_A) \wedge (f_A(x_A, \sigma) = \emptyset)]. \end{cases} \tag{12b}$$

Example 3 Let us consider the representative automaton A shown in Fig. 4a, where alphabet $\Sigma = \{a, b\}$ is ordered as $a \prec b$. For this automaton, we can obtain automaton $\phi(A)$ following a two step procedure: *(i)* we relabel the states of automaton A according to the function $\{(x_a, \sigma_k x_a) : x_a \in X_A\} = \{(\epsilon, b), (a, ba)\}$, add a new state labeled by ϵ (which will be the initial state of $\phi(A)$), therefore obtaining the set $X_\phi = \{\epsilon, b, ba\}$, and add the transition $(\epsilon, \sigma_k, \sigma_k) = (\epsilon, b, b)$ as depicted in Fig. 4b; *(ii)* we obtain $\phi(A)$ by adding all the necessary transitions of the form $(x_\phi, \sigma, \epsilon)$ until we have a complete automaton, as represented in Fig. 4c.

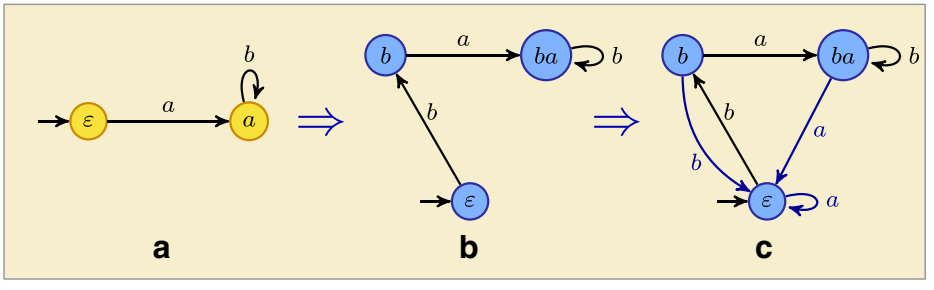


Fig. 4 Automaton $A \in \mathcal{A}^{(n,k)}$ (**a**); first step of the transformation of A into a complete automaton (**b**); complete automaton $\phi(A)$

The following lemma provides the foundation for the generation of the set $\mathcal{A}^{(n,k)}$ of accessible automata using the set $C^{(n+1,k)}$ of complete automata.

Lemma 1 (Bassino et al. 2009) *Let $\mathcal{E}^{(n+1,k)}$ be the subset of $\mathcal{C}^{(n+1,k)}$ formed with all the complete representative automata $E = (X_E, \Sigma, f_E, \Gamma_E, \epsilon)$ such that $f_E(\epsilon, \sigma) = \epsilon$ for $\sigma \in \Sigma \setminus \{\sigma_k\}$. Then, function ϕ is a bijection from $\mathcal{A}^{(n,k)}$ to $\mathcal{E}^{(n+1,k)}$.*

Remark 2 Notice that if $E \in \mathcal{E}^{(n+1,k)}$, then $\phi^{-1}(E)$ is obtained by deleting the initial state of E and its transitions, and turning the second state to the initial state in accordance with the lexicographical order.

It is important to stress that Lemma 1 transforms the problem of generating accessible automata with n states into the equivalent problem of generating complete automata with $n+1$ states. This is the basis for the formulation of Algorithm 6, which generates the set $\mathcal{A}^{(n,k)}$ of accessible, but not necessarily complete, automata. The correctness of Algorithm 6 is then ensured by the the fact that the representative automaton $E = (X_E, \Sigma, f_E, \Gamma_E, \epsilon) \in \mathcal{E}^{(n+1,k)}$ has the following representative sequence:

$$s_{f,E} = s_1 s_2 \dots s_{k(n+1)} = 0^{k-1} 1 s_k \dots s_{(n+1)k-1}. \tag{13}$$

Algorithm 6 Exhaustive generation of set $\mathcal{A}^{(n,k)}$ of accessible automata

STEP 1. Use Algorithm 3 to generate all the representative sequences corresponding to the $\mathcal{C}^{(n+1,k)}$ of complete automata and replace the final sequence s_f of Algorithm 3 (in line 4) with the sequence $s'_f = 0^{k-1}12\dots n\, n^{(k-1)n+1}$.

STEP 2. Transform each generated sequence into a complete automaton $G_c \in \mathcal{E}^{(n+1,k)} \subset \mathcal{C}^{(n+1,k)}$ and, for this automaton, delete the initial state (labeled by 0) and its transitions and define the state labeled by 1 as a new initial state. The resulting automaton, denoted by G, belongs to the set $\mathcal{A}^{(n,k)}$.

Due to the structure imposed by Eq. 13 to all representative sequences corresponding to automata in the set $\mathcal{E}^{(n+1,k)}$, the last sequence of flags that can be generated is $(f_j)_{j\in I_{1:n}} = (k+j-2)_{j\in I_{1:n}}$. This fact determines that the final sequence in Step 1 of Algorithm 6 must be $s'_f = 0^{k-1}12\dots n\, n^{(k-1)n+1}$.

Example 4 Let us now illustrate the use of Algorithm 6 to generate set $\mathcal{A}^{(2,2)}$ of accessible automata with $n = 2$ states and $k = 2$ events.

The first step is to generate the sequences of set $\mathcal{C}^{(n+1,k)} = \mathcal{C}^{(3,2)}$ (see Example 1) until the final sequence $s'_f = 0^{k-1}12\dots n\, n^{(k-1)n+1} = 012222$ is reached; this corresponds to the first 45 sequences listed in columns 1 and 2 of Table 1.

In order to illustrate the second step of Algorithm 1, consider sequence 012000 shown in Fig. 5a. This representative sequence corresponds to the complete automaton $G_c \in \mathcal{E}^{(3,2)} \subset \mathcal{C}^{(3,2)}$ depicted in Fig. 5b. Deleting the state labeled as 0 (together with all of its transitions) and defining the state labeled by 1 as the new initial state, we obtain the corresponding accessible automaton $G \in \mathcal{A}^{(2,2)}$ shown in Fig. 5c. We perform this step for all the 45 representative sequences obtained in Step 1, therefore obtaining the entire set $\mathcal{A}^{(2,2)}$.

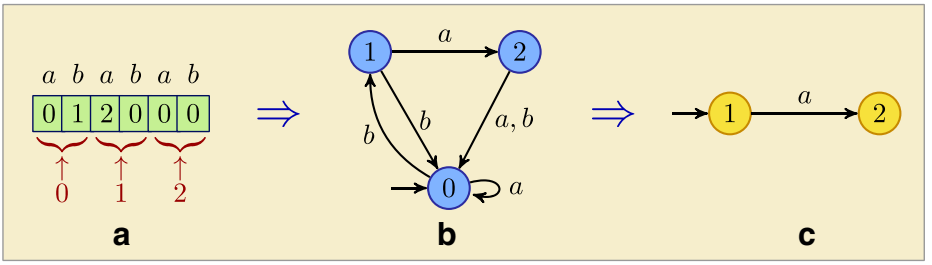


Fig. 5 Representative sequence of an automaton in $\mathcal{C}^{(3,2)}$ (**a**); corresponding complete automaton $G_c \in \mathcal{C}^{(3,2)}$ (**b**); accessible automaton $G \in A^{(2,2)}$ obtained from G_c (**c**)

As established in Lemma 1, the sets $\mathcal{A}^{(n,k)}$ and $\mathcal{E}^{(n,k)}$ have the same cardinality. This fact allow us to derive the following expression to compute the cardinality of $\mathcal{A}^{(n,k)}$:

$$\left|\mathcal{A}^{(n,k)}\right| = \sum_{h_1=k}^{2k-1} \sum_{h_2=h_1+1}^{3k-1} \sum_{h_3=h_2+1}^{4k-1} \cdots \sum_{h_{n-1}=h_{n-2}+1}^{kn-1} \prod_{i=1}^{n} (i+1)^{h_i-h_{i-1}-1}, \tag{14}$$

where $f_0 = k-1$ and $f_n = k(n+1)$. This equation was obtained by using the formula provided by Eq. 10 in Theorem 4. We observe that the leftmost summation in Eq. 10, which corresponds to the positional variation of the first flag f_1, must be removed since the structure of representative sequences imposed by Eq. 13 fixes f_1 to $k-1$. Then, the second leftmost summation of Eq. 10, which corresponds to the positional variation of flag f_2, must begin in k since the first $k-1$ positions of the representative sequence in Eq. 13 are fixed to $s_0 \ldots s_{k-1} = 0^{k-1}1$. The upper bound of the rightmost summation in Eq. 10 must be changed from $k(n-1)-1$ to $kn-1$ since we apply this formula for the set $\mathcal{E}^{(n+1,k)}$. Once again, since the first flag is held to $k-1$, we only need to count the combinatorics of numbers determined by the flags $f_2, f_3 \ldots, f_n$, which implies that the summation index must be changed from $i^{h_i-h_{i-1}-1}$ to $(i+1)^{h_i-h_{i-1}-1}$.

Let us now illustrate the use of Eq. 14 to compute the cardinality of $\mathcal{A}^{(2,2)}$. In this case,

$$\left|\mathcal{A}^{(2,2)}\right| = \sum_{f_1=2}^{3} \prod_{i=1}^{2} (i+1)^{f_i-f_{i-1}-1}\Big|_{f_0=1,\, f_2=6} = 2^0 \times 3^3 + 2 \times 3^2 = 45,$$

which is equal to the number of sequences obtained in Example 4, according to Algorithm 6.

3.3 Exhaustive experiment for the analysis of average state sizes of diagnosers and verifiers

Let Σ_k denote an event set with cardinality k formed with an ordered alphabet and assume that Σ_k is partitioned as $\Sigma_k = \Sigma_{k,o} \dot{\cup} \Sigma_{k,u}$, where $\Sigma_{k,o} = \{\sigma_1, \sigma_2, \ldots, \sigma_{k-u}\}$ and $\Sigma_{k,u} = \{\sigma_{k-u+1}, \sigma_{k-u+2}, \ldots, \sigma_k\}$, are the sets of observable and unobservable events, respectively. Note that $|\Sigma_{k,u}| = u$, i.e., the last u events of Σ_k are, by assumption, unobservable. Finally, assume that σ_k is the unique failure event.

Let

$$G_i^{(n,k,u)} = (X_n, \Sigma_k, f_i^{(n,k)}, \Gamma_i^{(n,k)}, 1) \in \mathcal{A}^{(n,k)},$$

where $X_n = \{1, \ldots, n\}$, denote an automaton whose event set Σ_k is partitioned as above; therefore having the last u ($1 \leq u < k$) events unobservable. The diagnoser (built in

accordance with Eq. 7) and the verifier (constructed by following the steps of Algorithm 2) will be denoted, respectively, as:

$$G_{D,i}^{(n,k,u)} = (X_{D,i}^{(n,k,u)}, \Sigma_{k,o}, f_{D,i}^{(n,k,u)}, \Gamma_{D,i}^{(n,k,u)}, x_{0D,i}^{(n,k,u)})$$

and

$$G_{V,i}^{(n,k,u)} = (X_{V,i}^{(n,k,u)}, \Sigma_{V,k}, f_{V,i}^{(n,k,u)}, \Gamma_{V,i}^{(n,k,u)}, (0N, 0N)).$$

Definition 6 (Set of valid automata) An automaton $G_i^{(n,k,u)} \in \mathcal{A}^{(n,k)}$ is a valid automaton for the experimental analysis of average state sizes of diagnosers and verifiers, if it satisfies the following conditions:

C1. The language generated by $G_i^{(n,k,u)}$ is live, i.e., $\Gamma_i^{(n,k)}(x) \neq \emptyset$ for all $x \in X_n$.

C2. For any $\sigma \in \Sigma_k$, there exists $x \in X_n$ such that $\sigma \in \Gamma_i^{(n,k)}(x)$, i.e., all events in Σ_k appear in $G_i^{(n,k,u)}$.

C3. $G_i^{(n,k,u)}$ does not have cyclic paths whose transitions are all labeled with unobservable events with, at least, one of these transitions, labeled with the failure event σ_k.

A set $\mathcal{A}_v^{(n,k,u)} \subset \mathcal{A}^{(n,k)}$ formed with those automata that satisfy conditions **C1**–**C3** is called a valid set of automata.

Although Condition **C1** is not necessary, it is usually made, without loss of generality, in the studies of failure diagnosability of DES. In our case, since all automata are generated automatically, property **C2** ensures that all automata have exactly k events, being exactly u unobservable; therefore including the failure event. Condition **C3** relaxes Assumption **A2** (Sampath et al. 1995), which precludes the existence of cycles of states connected with unobservable events only. According to Condition **C3**, one of the events in the cycle is the failure event, and, therefore, only those automata whose structure is known to be non-diagnosable a priori are being rejected. It is important to remark that, as far as diagnosability verification using the verifier automaton considered in this paper, not only Condition **C3** but also Assumption **A2** can be removed. In such case, Algorithm 2 does not fail to ascertain the language diagnosability.

We will consider the following class of valid automata:

$$\mathcal{A}_E = \{\mathcal{A}_v^{(n,k,u)} : (n, k, u) \in I_E\},$$

where,

$$\begin{aligned} I_E = \{&(3, 3, 1), (3, 3, 2), (3, 4, 1), (3, 4, 2), (3, 4, 3),\\ &(3, 5, 1), (4, 2, 1), (4, 3, 1), (4, 3, 2), (5, 2, 1)\} \end{aligned}$$

These sets have been chosen because, although the corresponding sets of valid automata have "moderate" cardinality, they are large enough to allow us to obtain significant statistic conclusions. The variables that will be used to compare the state sizes of automata $G_{D,i}^{(n,k,u)}$ and $G_{V,i}^{(n,k,u)}$ for the sets of valid automata $\mathcal{A}_v^{(n,k,u)}$, $(n, k, u) \in I_E$ are as follows:

1. $\mathbf{D}^{(n,k,u)} = (D_1^{(n,k,u)}, D_2^{(n,k,u)}, \dots) = (|X_{D,1}^{(n,k,u)}|, |X_{D,2}^{(n,k,u)}|, \dots)$: variable that contains the number of states of diagnoser $D_i^{(n,k,u)}$ for each automaton $G_i^{(n,k,u)}$ of the set of valid automata $\mathcal{A}_v^{(n,k,u)}$.

2. $\mathbf{V}^{(n,k,u)} = (V_1^{(n,k,u)}, V_2^{(n,k,u)}, \ldots) = (|X_{V,1}^{(n,k,u)}|, |X_{V,2}^{(n,k,u)}|, \ldots)$: variable that contains the number of states of verifier $V_i^{(n,k,u)}$ for each automaton $G_i^{(n,k,u)}$ of the set of valid automata $\mathcal{A}_v^{(n,k,u)}$.

The exhaustive experiment for the comparison of the state sizes of diagnosers and verifiers has been carried out according to the following steps:

Step 1: *Generations of set $\mathcal{A}_v^{(n,k,u)}$ of valid automata.*
This step is performed by combining the algorithm for exhaustive generation of accessible automata (Algorithm 6) with a rejection algorithm that excludes the automata that do not satisfy conditions **C1**–**C3**.

Step 2: *State size determination.*
In this step, for each set $\mathcal{A}_v^{(n,k,u)}$ of valid automata, taken from the database generated in Step 1, diagnosers and verifiers are constructed and the corresponding variables $\mathbf{D}^{(n,k,u)}$ and $\mathbf{V}^{(n,k,u)}$ with the values of their state space size stored for future statistical analysis.

Table 2 shows the results of the exhaustive experiment for the instances indexed by $(n, k, u) \in I_E$. Columns 2 and 3 show, respectively, the cardinality of sets $\mathcal{A}^{(n,k)}$ and $\mathcal{A}_v^{(n,k,u)}$ of accessible and valid automata. Columns 4–6 (resp. 9–10) show the maximum, the average, and the standard deviation values of variables $\mathbf{D}^{(n,k,u)}$ ($\mathbf{V}^{(n,k,u)}$), and column 7 (resp. 11) shows the upper bound in the number of states given by the worst case computational complexity, for diagnosers (resp. verifiers) construct for each automata $G_i^{(n,k,u)} \in \mathcal{A}_v^{(n,k,u)}$. For example, 23,846,125 accessible automata have been generated in order to form set $A^{(4,3)}$ but, among these automata, 4,577,395 valid automata have been selected to form $\mathcal{A}_v^{(4,3,1)}$. Notice that, for this set, the maximum and the average variable $\mathbf{D}^{(4,3,1)}$ ($\mathbf{V}^{(n,k,u)}$) are equal to 53 (resp. 20) and 8.7 (resp. 12.8), which means that, for this case, the average number of states of diagnosers is smaller than that of verifiers.

Comparing the worst case computational complexity, we can see from Table 2 that bound 2^{2n} has not been reached by any of the diagnosers and bound $2n^2$ was reached only once

Table 2 Results of the computational computation complexity analysis of diagnosers and verifiers for $\mathcal{A}_v^{(n,k,u)} \in \mathcal{A}_E$

	Number of automata		Diagnoser				Verifier			
(n, k, u)	$\lvert\mathcal{A}^{(n,k)}\rvert$	$\lvert\mathcal{A}_v^{(n,k,u)}\rvert$	max	m_D	s_D	2^{2n}	max	m_V	s_V	$2 \times n^2$
(3, 3, 1)	81856	18387	20	5.9	2.9	64	12	7.6	3.1	18
(3, 3, 2)	81856	7231	9	2.7	1.0	64	18	10.3	5.3	18
(3, 4, 1)	6516480	1472867	21	8.4	3.8	64	12	8.9	2.8	18
(3, 4, 2)	6516480	674727	20	4.8	2.4	64	18	12.4	5.0	18
(3, 4, 3)	6516480	280043	9	2.4	0.8	64	18	12.5	5.2	18
(3, 5, 1)	467590144	10830975	21	7.7	3.8	64	12	7.4	3.3	18
(4, 2, 1)	20225	3384	16	3.6	1.5	256	20	8.8	4.5	32
(4, 3, 1)	23846125	4577395	53	8.7	4.7	256	20	12.8	5.0	32
(4, 3, 2)	23846125	1201023	16	3.0	1.1	256	32	19.0	9.2	32
(5, 2, 1)	632700	90533	25	4.0	1.6	1024	30	12.2	6.4	50

(for $\mathcal{A}_v^{(4,3,2)}$) by the verifiers. In addition, notice that for all sets $\mathcal{A}_v^{(n,k,u)}$, $(n,k,u) \in I_E$, the average state size of diagnosers is lower than the worst case computational complexity; for some cases is even lower by several orders of magnitude. On the other hand, the average state size of verifiers has the same order of magnitude as its worst case complexity.

It is interesting to remark that the average state size of diagnosers (resp. verifiers) decreases (resp. increases) monotonically with an increase in the number of unobservable events. For instance, in the sets $\mathcal{A}_v^{(3,4,1)}$, $\mathcal{A}_v^{(3,4,2)}$ and $\mathcal{A}_v^{(3,4,3)}$ of valid automata, whose numbers of unobservable events increases from 1 to 3, the average state size of diagnosers (verifiers) decreases (increases) from 8.4 (resp. 8.9) to 2.4 (resp. 12.5). Another interesting fact is that variables $\mathbf{V}^{(3,3,2)}$, $\mathbf{V}^{(3,4,2)}$ and $\mathbf{V}^{(3,4,3)}$ achieve the worst case complexity of $2 \times n^2 = 2 \times 3^2 = 18$, and, also, $\mathbf{V}^{(4,3,2)}$, which has worst case complexity of $2 \times 4^2 = 32$.

In summary, the more relevant results of this preliminary study on the average state sizes of diagnosers and verifiers carried out through exhaustive experimentation on the set $\mathcal{A}^{(n,k,u)}$, for $(n,k,u) \in I_E$ are as follows: *(i)* the worst case bound 2^{2n} in the state size of diagnosers is unlike to be reached; *(ii)* the average state size of verifiers is of the same order of magnitude as its worst case bound, i.e., $2n^2$ is also an appropriate bound for the average state size of verifiers.

Notice that these conclusions are restricted to the small classes of sets of automata described by $\mathcal{A}_v^{(n,k,u)}$, $(n,k,u) \in I_E$. In order to further extend this study and to obtain significant practical results, it is necessary to consider larger sets of accessible automata $\mathcal{A}^{(n,k)}$ and to formulate quantitative hypothesis regarding the expected values of the size of diagnosers and verifiers. However, such expansions on the sizes of automata is seriously hampered by the number of automata in sets $\mathcal{A}^{(n,k)}$ when n increases, as shown in Eq. 14. Exhaustive experiments could be extended further for larger values of n by using supercomputers together with fine tuned programs in a compiled language. However, the scope of such an approach would still be limited; for example, a computational improvement of order 10^6 (a program 1000 times faster running in parallel in 1000 computers) is still a tiny improvement in the task of testing set $\mathcal{A}^{(10,5)}$, whose cardinality is approximately 2.6×10^{47}.

In the next section, we will propose a new approach for the problem of performing the computation of the number of states of diagnosers and verifiers for instances larger than those in I_E which consists of replacing the whole set of automata in $\mathcal{A}^{(n,k)}$ with a uniform sample of automata in $\mathcal{A}^{(n,k)}$.

4 Experimental analysis of average computational complexity of failure diagnosis of DES using sampling

4.1 Uniform random generation of accessible automata

In this section we will present the method proposed by Bassino and Nicaud (2007) and Bassino et al. (2009) for the uniform random generation of accessible automata. We start with the following definition.

Definition 7 (**k-Dick partition**) Let $\mathcal{Q}^{(p,n)}$ denote the class of partitions of set $I_{1:p}$ in n blocks and assume that $Q = \{Q_0, \ldots, Q_{n-1}\}$ is an element of $\mathcal{Q}^{(p,n)}$, where the elements of Q are ordered according to their smallest element. For a given $k \geq 2$, a partition $Q \in \mathcal{Q}^{(p,n)}$ is a k-Dick if $\min Q_j \leq kj + 1$, $j = 0, 1, 2, \ldots, n-1$.

For example, consider partitions $Q' = \{Q'_0, Q'_1\}$ and $Q'' = \{Q''_0, Q''_1\}$ of set $I_{1:5}$, where $Q'_0 = \{1, 4, 5\}$, $Q'_1 = \{2, 3\}$, $Q''_0 = \{1, 2, 3\}$, and $Q''_1 = \{4, 5\}$. Notice that Q' is a 2-Dick partition since $\min Q'_0 = 1 \leq 2 \times 0 + 1 = 1$ and $\min Q'_1 = 2 \leq 2 \times 1 + 1 = 3$. However, Q'' is not a 2-Dick partition since $\min Q''_1 = 4 > 2 \times 1 + 1 = 3$.

Let us now consider a complete accessible representative automaton $G'_{ca} = (X', \Sigma, f', \Gamma', \epsilon) \in \mathcal{C}^{(n,k)}$ and let $T = \left\{(x', \sigma, f(x', \sigma)) : x' \in X', \sigma \in \Sigma\right\} \cup \{(\emptyset, \epsilon, \epsilon)\}$ the set of all transitions of G'_{ca}, including the additional transition $(\emptyset, \epsilon, \epsilon)$ (which can be interpreted as the arrow that points to the initial state in the state diagram of G'_{ca}). Notice that $|T| = kn + 1$ since G'_{ca} is complete. It is then possible to establish bijection $\nu : T \to I_{1:kn+1}$, defined as follows:

B1. $\nu((\emptyset, \epsilon, \epsilon)) = 1$,
B2. $\nu((\epsilon, \sigma_1, \sigma_1)) = 2$,
B3. For each pair of transitions (r, σ_a, s) and (t, σ_b, u) in $T \setminus \{(\emptyset, \epsilon, \epsilon)\}$, then $\nu((r, \sigma_a, s)) < \nu((t, \sigma_b, u))$ if and only if $r\sigma_a <_{lex} t\sigma_b$.

The above defined bijection can be illustrated with the help of the representative automaton shown in Fig. 6a. Automaton G'_{ca} with transitions numbered according to bijection ν is depicted in Fig. 6b. Notice that the arrow that points to the initial state has been labeled as 1 (rule **B1**), $\nu((\epsilon, a, a)) = 2$ (rule **B2**) and that, according to rule **B3**, $\nu((aab, a, aa)) = 6 < \nu((aab, b, aab)) = 7$, since $aaba <_{lex} aabb$. It is worth remarking that the order of the transitions defined by ν is unique and corresponds to the order in which the transitions of G'_{ca} are traversed in a depth-first search algorithm according to a lexicographical order.

After obtaining a bijection that maps the transitions of G'_{ca} into integers in $I_{1:kn+1}$, we will show how G'_{ca} can be expressed in terms of a k-Dick partition of $I_{1:kn+1}$ formed with n sets. In order to do so, let $Q_D = \{Q_0, \ldots, Q_{n-1}\}$ denote a partition built from G'_{ca} according to the following rules:

E1. $1 \in Q_0$
E2. If i and j are, respectively, the mapping of transitions

$$\nu((r, \sigma_a, s)), \nu((t, \sigma_b, u)) \in T \setminus \{(\emptyset, \epsilon, \epsilon)\},$$

then i, j are in the same element of Q_D if, and only if, $s = u$.

The following theorem establishes the bijection between the set of complete accessible automata and k-Dick partitions.

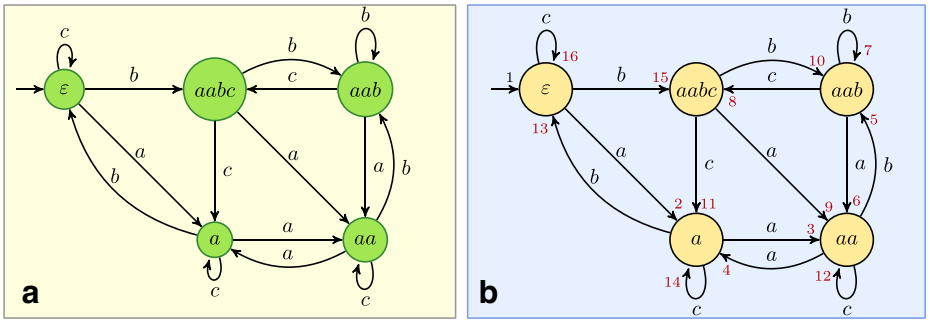


Fig. 6 Representative automaton $G'_{ca} = (X, \Sigma, f, \Gamma, x_0)$ (**a**); G'_{ca} with transitions numbered in accordance with bijection ν (**b**)

Theorem 5 *Bassino et al.* (2009) *Let $\mathcal{C}^{(n,k)}$ denote the set of complete accessible automata with n states and k events and $\mathcal{Q}^{(kn+1,n)}$ the class of partitions of $I_{1:kn+1}$ in n blocks. Let $\Omega : \mathcal{C}^{(n,k)} \rightarrow \mathcal{Q}^{(kn+1,n)}$ denote a mapping where $Q_D = \Omega(G'_{ca})$ is a partition built in accordance with rules* **E1** *and* **E2**. *For each $n \geq 1$ and $k \geq 2$, Ω is a bijection between $\mathcal{C}^{(n,k)}$ and the set of k-Dick partitions in $\mathcal{Q}^{(kn+1,n)}$.*

Let us illustrate the construction of the partition associated with a complete accessible automaton with the help of automaton G'_{ca} depicted in Fig. 6. Rule **E1** imposes that $Q_D = \{\{1, \ldots\}, Q_1, \ldots, Q_4\}$, since $n = 5$. Rule **E2** implies that all transition numbers that arrive to the same state must belong to the same element of Q_D. Therefore, the following partition corresponds to G'_{ca}:

$$Q_D = \{\{1, 13, 16\}, \{2, 4, 11, 14\}, \{3, 6, 9, 12\}, \{5, 7, 10\}, \{8, 15\}\}.$$

Notice that, the objective here is to generate uniformly distributed accessible automata within the set $\mathcal{A}^{(n,k)}$. In order to do so, we must consider the following relevant facts: *(i)* Lemma 1 changes the problem of generating accessible automata with n states in a problem of generating complete automata within the set $\mathcal{C}^{(n+1,k)}$; *(ii)* Theorem 5 converts the problem of generating complete automata in $\mathcal{C}^{(n+1,k)}$ to a problem of generating k-Dick partitions of the set $I_{1:kn+1}$ with $n + 1$ non-empty blocks.

The problem of generating uniformly distributed accessible automata in the set $\mathcal{A}^{(n,k)}$ has been addressed by Bassino and Nicaud (2007) and Bassino et al. (2009). We will present in the sequel the three algorithms necessary for the generation of a k-Dick partition $Q_D \in \mathcal{Q}^{(k(n+1)+1,n+1)}$ such that $\Omega^{-1}(Q_D) \in \mathcal{E}^{(n+1,k)}$ and its transformation into and equivalent accessible automaton. The first algorithm (Algorithm 7) deploys Boltzmann sampler method to generate the structure of a partition $Q \in \mathcal{Q}^{(p,n)}$. The second algorithm uniformly generates a k-Dick partition $Q_D \in \mathcal{Q}^{(k(n+1)+1,n+1)}$ such that $\Omega^{-1}(Q_D) \in \mathcal{E}^{(n+1,k)}$. Finally, Algorithm 9 converts the partition generated in Algorithm 7 to an automaton $A \in \mathcal{A}^{(n,k)}$. The interested reader is referred to the works by Bassino and Nicaud (2007) and Bassino et al. (2009) for the theoretical justification of the algorithms. The example that follows illustrates the application of Algorithms 7, 8 and 9 to the uniform generation of a 2-Dick partition.

Algorithm 7 Generation of the structure of a partition $Q \in \mathcal{Q}^{(p,q)}$

Inputs: Integers p and q

Output: Structure R of a partition in the set $\mathcal{Q}^{(p,q)}$.

STEP 1: Compute λ by solving the following equation

$$\lambda - \frac{p}{q}(1 - e^{-\lambda}) = 0. \tag{15}$$

STEP 2: Generate a vector of integers $\mathbf{r} = (r_1, \ldots, r_q)$ using the shifted Poisson distribution $\text{Poisson}_{\geq 1}(\lambda)$.

STEP 3: If $\sum_{i=1}^{q} r_i \neq p$ go back to Step 2. Otherwise form set R as follows:

$$R = \{\{1, \ldots, r_1\}, \{r_1 + 1, \ldots, r_1 + r_2\}, \ldots, \{r_1 + \cdots + r_{q-1} + 1, \ldots, p\}\}.$$
$$= \{R_1, R_2, \ldots, R_q\}$$

Algorithm 8 Uniform generation of a k-Dick partition of $\mathcal{Q}^{(k(n+1)+1,n+1)}$ such that $\Omega^{-1}(Q_D) \in \mathcal{E}^{(n+1,k)}$

Inputs: number of states and events, n and k, respectively, and a structure R of a partition in the set $Q \in \mathcal{Q}^{(kn+1,n+1)}$.

Output: a k-Dick partition $Q_D \in \mathcal{Q}^{(k(n+1)+1,n+1)}$ such that $\Omega^{-1}(Q_D) \in \mathcal{E}^{(n+1,k)}$.

STEP 1: Use Algorithm 7 to obtain a structure of a partition in $\mathcal{Q}^{(kn+1,n+1)}$.

STEP 2: Generate a uniformly distributed bijection $\delta : I_{1:kn+1} \to I_{k+1:k(n+1)} \cup \{1\}$.

STEP 3: Form set $Q_D = \left\{\{\delta(i) : i \in R_j\} : R_j \in R,\, j \in I_{1:n+1}\right\}$.

STEP 4: Define $Q_D = \{Q_{D,0}, \ldots, Q_{D,n}\}$ in such a way that sets $Q_{D,0}, \ldots, Q_{D,n}$ are ordered in accordance with the smallest element and $Q_{D,0} \leftarrow Q_{D,0} \cup I_{2:k}$.

STEP 5: Choose uniformly an integer $j \in I_{0:n}$ and set $Q_{D,j} \leftarrow Q_{D,j} \cup \{k(n+1)+1\}$.

STEP 6: If Q_D, obtained in Step 5, is not a k-Dick, go back to Step 1. Otherwise terminate the algorithm.

Algorithm 9 Transformation of a k-Dick partition into an accessible automaton

Input: *A* k-Dick partition of $\mathcal{Q}^{(k(n+1)+1,n+1)}$ such that $\Omega^{-1}(Q_D) \in \mathcal{E}^{(n+1,k)}$, and an ordered alphabet $\Sigma_A = \{\sigma_1, \ldots, \sigma_k\}$, where $\sigma_1 \prec \sigma_2 \prec \ldots \prec \sigma_n$

Output: *An accessible automaton* $A = (X_A, \Sigma, f_A, \Gamma_A, \mathbb{1})$ STACK $S \longleftarrow \emptyset$

$T \longleftarrow \emptyset$

$q_{lex} \longleftarrow 2$ *(second state of the automaton according to a lexicographical order, being 1 the initial state)*

for $\sigma' \in \Sigma$ *traversed in inverse order* **do**

  PUSH$(S, (1, \sigma'))$

for $i \in I_{k+2:k(n+1)+1}$ **do**

  $(p, \sigma) \longleftarrow$ POP(S)

  $q = j$, where j is the label associated with set $Q_{D,j}$ for which $i \in Q_{D,j}$

  if $q = q_{lex}$ **then**

    $q_{lex} \longleftarrow q_{lex} + 1$

    for $\sigma' \in \Sigma$ *traversed in inverse order* **do**

      PUSH$(S, (q, \sigma'))$

  if $q \neq 0$ **then**

    $T \longleftarrow T \cup \{(p, \sigma, q)\}$

End Set $A = (X_A, \Sigma, f_A, \Gamma_A, 1)$, where $X_A = \{1, \ldots, n\}$, $\Sigma_A = \Sigma$ and f_a and Γ_A are obtained from the transitions defined in T.

Example 5 Assume that we want to generate an automaton in $\mathcal{A}^{(2,2)}$ with uniform distribution. In this case, $n = k = 2$, and, thus, according to Algorithm 9, it is necessary to generate a 2-Dick partition of $\mathcal{Q}^{(7,3)}$ with uniform distribution. Such a partition can be obtained by following the steps of Algorithm 8, whose first step is to use Algorithm 7 to generate a structure of a partition in $\mathcal{Q}^{(5,3)}$, as follows:

1. Solving equation $\lambda - \frac{5}{3}(1 - e^{-\lambda}) = 0$, we obtain $\lambda = 1.1262$.
2. We use the shifted Poisson distribution $\text{Poisson}_{>1}(1.1262)$ to generate a partition structure $\mathbf{r} = (r_1, r_2, r_3)$.

3. Assume that a first run returns $\mathbf{r} = (1, 2, 1)$. Since $\sum_{i=1}^{3} r_i = 4 \neq p = 5$, another vector must be generated.
4. Assume that after a few runs, vector $\mathbf{r} = (1, 2, 2)$ is generated. Since $\sum_{i=1}^{3} r_i = 5$, the following partition structure can be returned from Algorithm 7:

$$R = \{\{1\}, \{2, 3\}, \{4, 5\}\}.$$

Once a partition structure has been generated, the next steps of Algorithm 8 must be executed, as follows:

1. Step 2: assume that the following uniformly distributed random bijection $\delta : I_{1:5} \rightarrow \{1, 3, 4, 5, 6\}$ has been generated:

$$\delta = \{(1, 5), (2, 3), (3, 4), (4, 6), (5, 1)\}.$$

2. Step 3: using the bijection obtained in the previous step, we replace the elements of $R_j \in R$, $j = 1, 2, 3$ with $\delta(i)$, and obtain $Q_D = \{\{5\}, \{3, 4\}, \{6, 1\}\}$.
3. Step 4: we must order Q_D according to the smallest elements and, in the sequel, redefine $Q_{D,0} \leftarrow Q_{D,0} \cup \{2\}$. Proceeding in this way, we obtain $Q_D = \{Q_{D,0}, Q_{D,1}, Q_{D,2}\} = \{\{1, 2, 6\}, \{3, 4\}, \{5\}\}$.
4. Step 5: a random number $j \in \{0, 1, 2\}$ must be uniformly chosen. Assume that $j = 1$ has been drawn. Therefore, $Q_{D,1} \leftarrow Q_{D,1} \cup \{k(n + 1) + 1\} = Q_{D,1} \cup \{7\}$, and so, $Q_D = \{\{1, 2, 6\}, \{3, 4, 7\}, \{5\}\}$.
5. Step 6: It can be easily checked the Q_D is a 2-Dick partition, and, thus, Algorithm 8 comes to an end.

We will now perform the steps of Algorithm 9 to transform partition $Q_D = \{\{1, 2, 6\}, \{3, 4, 7\}, \{5\}\}$ in an automaton $A = (I_{1:2}, \{a, b\}, f_A, \Gamma_A, 1) \in \mathcal{A}^{(2,2)}$. Notice that the algorithm variables are as follows:

– S: stack (initially empty);
– T: set of the transitions of automaton A (initially empty);
– q_{lex}: index of the next state to be visited according to the used lexicographical order (initialized in 2, since 1 is the initial state);
– (p, σ, q): candidate transition to be added.

The loop defined in lines 4 and 5 of the algorithm initializes the stack with ordered pairs of form $(1, \sigma')$, where the first entry is the initial state and the second one is an event of the alphabet, as follows:

$$S = \boxed{\begin{array}{c}(1, a)\\ \hline (1, b)\end{array}}.$$

Note that the pairs are placed in the stack in reverse lexicographical order. The main loop (lines 6–14) iterates on the elements of set $k + 2 : k(n + 1) + 1 = \{4, 5, 6, 7\}$, which enumerates all possible transitions of the accessible automaton that is being constructed. Table 3 shows the values of all variables for each value of $i \in \{4, 5, 6, 7\}$ at the end of each iteration; for example, the first row of Table 3 shows the values the variables for $i = 4$ that have been obtained as follows: *(i)* pair $(p, \sigma) = (1, a)$ is removed from S (line 7); *(ii)* since $i = 4 \in Q_{D,1}$, the output state of the first transition is $q = 1$ (line 8); *(iii)* since $q \neq q_{lex}$, lines 10–12 are not executed; *(iv)* since $q \neq 0$ (checked in line 13), transition $(1, a, 1)$ is added to T (line 14). Finally, when $i = 7$, the transition set $T =$

Table 3 Execution of Algorithm 9 to convert partition $Q_D = \{\{1, 2, 6\}, \{3, 4, 7\}, \{5\}\}$ into an accessible automaton

	q_{lex}	(p, σ, q)	S	T
i = 4	2	$(1, a, 1)$	$\boxed{(1, b)}$	$\{(1, a, 1)\}$
i = 5	3	$(1, b, 2)$	$\boxed{(2, a)}$ $\boxed{(2, b)}$	$\{(1, a, 1), (1, b, 2)\}$
i = 6	3	$(2, a, 0)$	$\boxed{(2, b)}$	$\{(1, a, 1), (1, b, 2)\}$
i = 7	3	$(2, b, 1)$	$\boxed{}$	$\{(1, a, 1), (1, b, 2), (2, b, 1)\}$

$\{(1, a, 1), (1, b, 2), (2, b, 1)\}$ is obtained, leading to the automaton whose state transition diagram is depicted in Fig. 7.

4.2 Evaluating the sample size of the experiment

In this subsection, we will determine the sample size necessary to draw significant statistical conclusions regarding the average state sizes of diagnosers and verifiers within a specified confidence interval based on two fundamental results of probability theory: the strong law of large numbers and the central limit theorem.

Let us consider the sample $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_N)$ of size N such that $E[Y_i] = m$ and $\text{var}[Y_i] = s^2$, where $E[\cdot]$ and $var[\cdot]$ denote, respectively, expected value and variance. The strong law of the large numbers ensures that the sample mean

$$\overline{M}_N = \frac{1}{N} \sum_{i=1}^{N} Y_i$$

is a consistent estimator for the value of m. If $E_N = \overline{M}_N - m$ is defined as the random variable that represents the error in the estimation of m, then the central limit theorem establishes that $E_N \approx N(0, s^2/N)$, i.e., the variable that represents the error in the estimation of m has an approximately normal distribution (Gubner 2006). Based on these facts, the confidence interval for m will be given as:

$$\left[\overline{M}_N - \frac{s z_{1-\alpha/2}}{\sqrt{N}}, \overline{M}_N + \frac{s z_{1-\alpha/2}}{\sqrt{N}}\right], \tag{16}$$

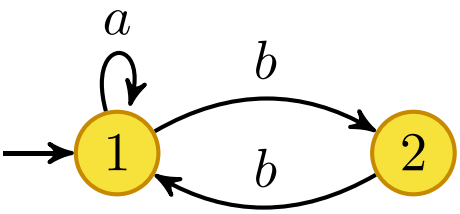


Fig. 7 Accessible automaton $A \in \mathcal{A}^{(2,2)}$ corresponding to partition $Q_D = \{\{1, 2, 6\}, \{3, 4, 7\}, \{5\}\}$, obtained according to Algorithm 9

Table 4 Maximum percent error in the estimation of mean ρ of variables $\mathbf{D}^{(n,k,u)}$ and $\mathbf{V}^{(n,k,u)}$ for a sample size $N = 10,000$

(n, k, u)	$\rho_{max,V}$	$\rho_{max,D}$
(3, 3, 1)	0.0080	0.0097
(3, 4, 1)	0.0063	0.0088
(3, 4, 2)	0.0080	0.0096
(3, 4, 3)	0.0082	0.0068
(3, 5, 1)	0.0089	0.0098
(4, 3, 1)	0.0076	0.0108
(4, 3, 2)	0.0096	0.0071
(5, 2, 1)	0.0104	0.0078

where $z_{1-\alpha/2}$ is the value of z for which $\phi(z) = 1 - \alpha/2$, being $\phi(z)$ the standard normal continuous density function, i.e., $\phi(z) = \frac{1}{2\pi}\int_{-\infty}^{z} e^{\frac{-t^2}{2}} dt$. Based on that, we can define the percent error in the estimation of m as follows:

$$\rho = \frac{s z_{1-\alpha/2}}{\overline{M}_N \sqrt{N}}. \tag{17}$$

We can now evaluate the sample size to be used in the experiments for the estimation of the average state sizes of diagnosers and verifiers. Consider the following specifications: *(i)* confidence level of 95%, *(ii)* percent error $\rho = 1\%$. Specification *(i)* implies that $1 - \alpha = 0.95$ and thus $\alpha = 0.05$ and $z_{1-\alpha/2} = 1.96$. The exact values of the mean and of the standard deviation, m and s, respectively, were determined exactly for state sizes of verifiers and diagnosers in the exhaustive experiments carried out in Section 3.3 (see Table 2). Based on those values, we will now check if the sample size $N = 10,000$ guarantees a confidence interval that also satisfies specifications *(i)* and *(ii)*.

Columns 2 and 3 of Table 4 show the maximum percent error (calculated using Eq. 17 with the exact value for the mean of each variable,, i.e., with $\overline{M}_N = m$) in the estimation of the mean for the variables $\mathbf{D}^{(n,k,u)}$ and $\mathbf{V}^{(n,k,u)}$, respectively, that represent the average state sizes of diagnosers and verifiers for the valid sets of automata with n states, k events, and u unobservable events. Notice that for all variables, the maximum error in the estimation of m is close to 1%. Therefore, $N = 10,000$ represents a reliable sample size for an experiment with uniform sample based on the partial knowledge provided by the exhaustive experiment.

4.2.1 Experimental procedure

Let us denote $\hat{\mathcal{A}}_v^{(n,k,u)}$ the set formed with $N = 10,000$ valid automata uniformly generated according to Algorithms 8 and 9. For the experimental analysis carried out here, the sets of uniformly generated valid automata belonging to the following class have been considered:

$$\mathcal{A}_U = \{\hat{\mathcal{A}}_v^{(n,k,u)} : (n, k, u) \in I_U\},$$

where

$$\begin{aligned}
I_U &= I_1 \cup I_2, \cup I_3, \\
I_1 &= \{(n, k, u) : n \in I_{3:20}, k \in I_{2:10}, u \in I_{1:k-1}\}, \\
I_2 &= \{(n, k, u) : n \in I_{3:20}, k \in \{16, 24\}, u \in I_{1:k-1}\}, \\
I_3 &= \{(n, k, u) : n \in \{32, 64\}, k \in I_{2:10} \cup \{16, 24\}, u \in I_{1:k-1}\}.
\end{aligned}$$

It can be easily checked that 16,600,000 automata have been used in the experiments, since $|A_U| = 1660$, i.e., 1660 sets of uniformly generated valid automata have been created.

For each set $\hat{\mathcal{A}}_v^{(n,k,u)}$, $(n, k, u) \in I_U$, the following variables have been considered in the experiments:

1. $\hat{\mathbf{D}}^{(n,k,u)} = (D_1^{(n,k,u)}, D_2^{(n,k,u)}, \ldots, D_{10000}^{(n,k,u)})$: variable with 10,000 components, where each component represents the cardinality of the set of states of the diagnoser associated with automaton $G_i^{(n,k,u)} \in \hat{\mathcal{A}}_v^{(n,k,u)}$, i.e., $D_i^{(n,k,u)} = |X_{D,i}^{(n,k,u)}|$.
2. $\hat{\mathbf{V}}^{(n,k,u)} = (V_1^{(n,k,u)}, V_2^{(n,k,u)}, \ldots, V_{10000}^{(n,k,u)})$: variable with 10,000 components, where each component represents the cardinality of the set of states of the verifier associated with automaton $G_i^{(n,k,u)} \in \hat{\mathcal{A}}_v^{(n,k,u)}$, i.e., $V_i^{(n,k,u)} = |X_{V,i}^{(n,k,u)}|$.

The experiments have been performed in two steps,

Step 1: *Generation of set $\hat{\mathcal{A}}_v^{(n,k,u)}$ of uniformly generated valid automata:*
This step is performed by combining the uniform generator of automata described in Section 4.1 with a rejection algorithm. For each automaton generated, the set of observable events $\Sigma_{k,o}$ and the set of failure event $\Sigma_f = \{\sigma_k\}$ are defined, leading to a candidate automaton $G_i^{(n,k,u)}$. In the sequel, the rejection algorithm checks if $G_i^{(n,k,u)}$ satisfies Properties **C1**–**C3**, and if it does so, $G_i^{(n,k,u)}$ will be incorporated to set $\hat{\mathcal{A}}_v^{(n,k,u)}$. Otherwise another automaton is generated. The process continues until 10,000 valid automata are generated.

Step 2: *Computation of state sizes*
In this step, a diagnoser and a verifier are constructed for each automaton $G_i^{(n,k,u)} \in \hat{\mathcal{A}}_v^{(n,k,u)}$, forming variables $\hat{\mathbf{D}}^{(n,k,u)}$ and $\hat{\mathbf{V}}^{(n,k,u)}$.

4.3 Data exploratory analysis

We will now present the statistical analysis based on the experimental results of variables $\hat{\mathbf{D}}^{(n,k,u)}$ and $\hat{\mathbf{V}}^{(n,k,u)}$, $(n, k, u) \in I_1$. Notice that, since $|I_1| = 810$ and 10,000 is the sample size, 8,100,000 diagnosers and 8,100,000 verifiers have been computed.

4.3.1 Diagnoser average state size analysis

Figure 8a and b show in logarithmic scale (base 2),[5] the average state size and the standard deviation of the state sizes of the diagnosers computed in the experiment for class I_1. The abscissa and ordinate represent, respectively, the number of states and events of G and the colormap, the number of unobservable events. The most relevant observations that can be made from the plots are as follows:

– The average state size of diagnosers has a monotonic sub-exponential behavior. Such information can be extracted directly from Fig. 8a, since the plot has logarithmic scale.
– For a class formed with sets of automata $\hat{\mathcal{A}}_v^{(n_0,k_0,u)}$, where n_0 and k_0 are fixed, and differ only in the number $u \in I_{0:k-1}$ of unobservable events, the largest average state size is obtained for $\hat{\mathcal{A}}_v^{(n_0,k_0,1)}$, i.e, for the set of automata with only one unobservable event.

[5] Base two has been chosen since the worst case computational complexity for diagnoser is $\mathcal{O}(2^n)$.

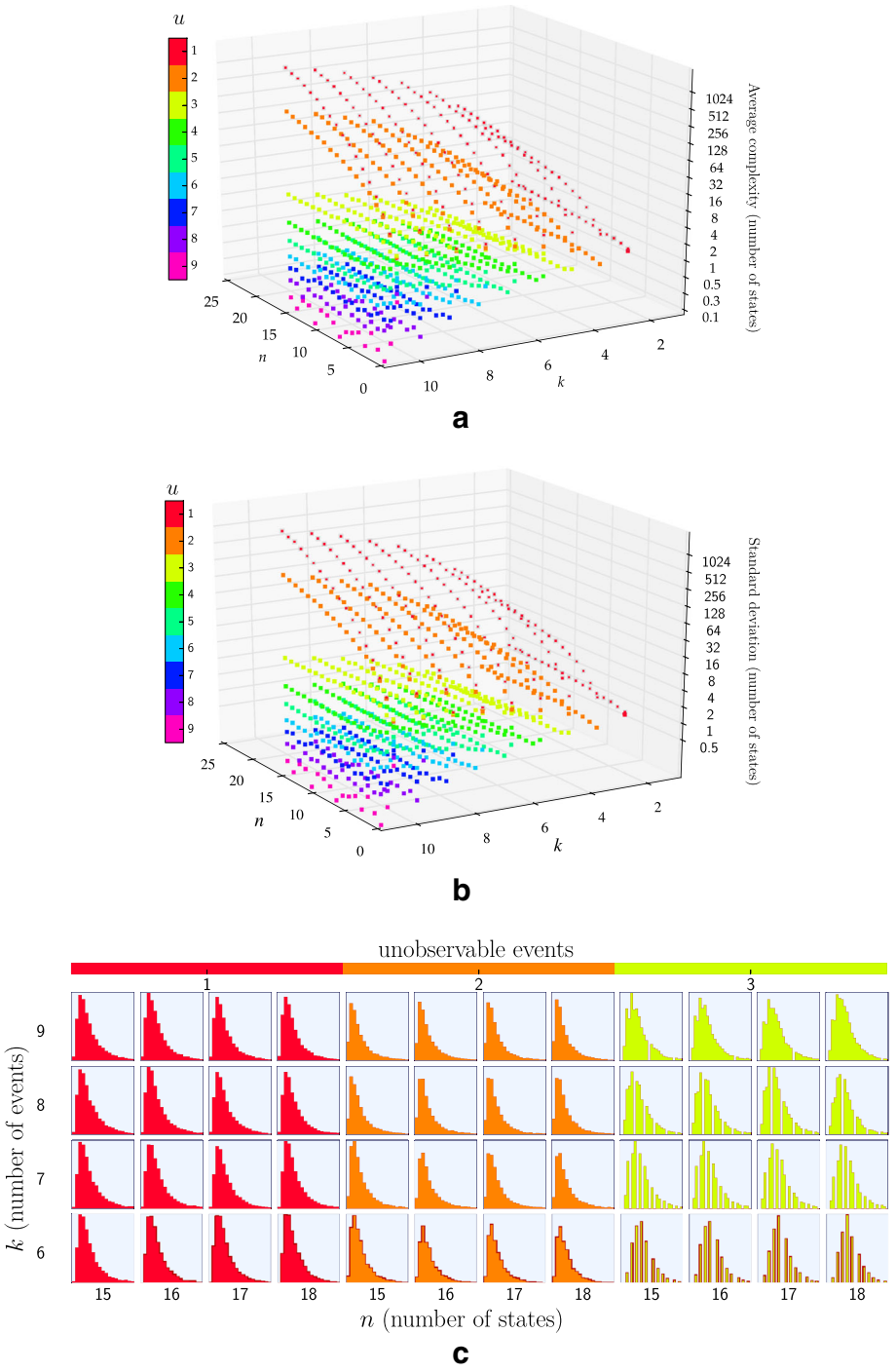


Fig. 8 Average of observed variables $\hat{\mathbf{D}}^{(n,k,u)}$, $(n,k,u) \in I_1$ (**a**); Standard deviations of variables $\hat{\mathbf{D}}^{(n,k,u)}$, $(n,k,u) \in I_1$ (**b**); Histograms of variables $\hat{\mathbf{D}}^{(n,k,u)}$, $n \in I_{15:18}$, $k \in I_{6:9}$, $u \in \{1,2,3\}$ (**c**)

Such a conclusion is consistent with Jirásková and Masopust (2012), which shows that the worst case computational complexity of natural projections of language occurs for automata with least transitions labeled with unobservable events. On the other hand, the smallest average state size is obtained for $\hat{\mathcal{A}}_v^{(n_0,k_0,k-1)}$, i.e., when there is only one observable event. Notice that the average state size decreases monotonically with the increase of unobservable events.

As we can observe in Fig. 8b, the standard deviations of variables $\hat{\mathbf{D}}^{(n,k,u)}$, $(n,k,u) \in I_1$ have a similar behavior as that of the averages discussed above.

Figure 8c shows the histograms for variables $\hat{\mathbf{D}}^{(n,k,u)}$, $n \in I_{15:18}$, $k \in I_{6:9}$, $u = 1, 2, 3$, where all plots in the same row (resp. column) have the same number of events (resp. states), and the colors identify different numbers of unobservable events. It can be seen from the plots that it is possible to establish a lognormal pattern for the distribution of state size of diagnosers, similar to that obtained when exhaustive experiments were carried out (Clavijo 2014).

4.3.2 Verifier average state size analysis

Figure 9a shows, in squared scale in the number of states,[6] the average state size of the verifiers computed in the experiment for class I_1. From the plot, the most important observations that can be made are as follows:

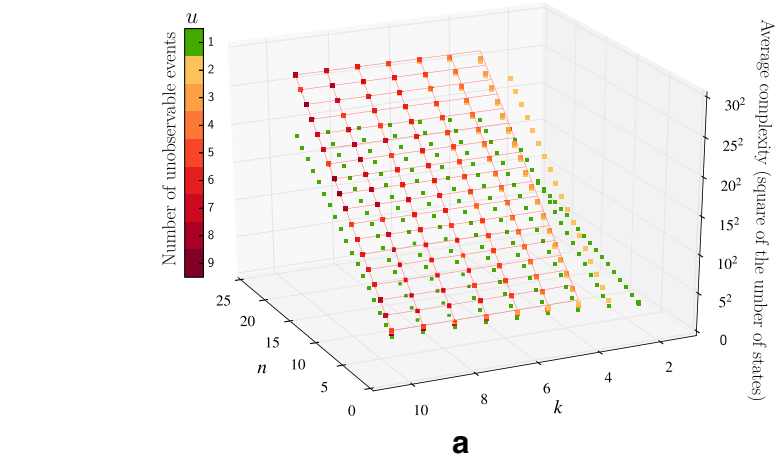
O1. Due to vertical axis square scale, the average state size of verifiers is quadratic. In order to further highlight this conclusion the plane $2n^2$ has been represented over the points corresponding to the sample means.

O2. For a class formed with sets of valid automata $\hat{\mathcal{A}}^{(n_0,k_0,u)}$, where $n_0 \in I_{3:20}$ and $k_0 \in I_{2:10}$ are fixed, differing only in the number of unobservable events, the smallest average state size of verifiers appears in the set $\hat{\mathcal{A}}^{(n_0,k_0,1)}$, i.e., with only one unobservable event.

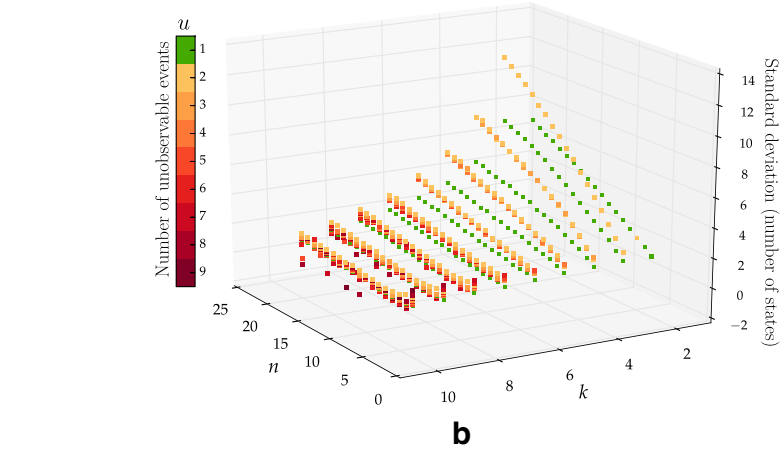
O3. For $k_0 \geq 4$, the average state size of verifiers for sets $\hat{\mathcal{A}}^{(n,k_0,u)}$, $u \in I_{2:10}$, is approximately equal to $2n^2$.

Figure 9b shows the standard deviations of variables $\hat{\mathbf{V}}^{(n,k,u)}$, $(n,k,u) \in I_1$. The most important conclusion that can be drawn from this plot is that the standard deviation decreases with an increase in the number of events; for example, for $k = 10$, the standard deviation is approximately equal to 4 (states), for any number n of states. Such a fact corroborates observation (**O3**) in the sense that not only the average state size of verifiers approximates $2n^2$ but also most of the verifiers constructed has $2n^2$ states. Finally, Fig. 9c shows the histograms of variables $\hat{\mathbf{V}}^{(n,k,u)}$, $n \in I_{11:14}$, $k \in \{4, 5, 8, 10\}$, $u = 1, 2, 3$, where all plots in the same row (resp. column) have the same number of events (resp. states), and the colors identify different number of unobservable events. The abscissas of each histogram ranges from 0 to $2n^2$. A closer look at the diagram also reveals that the state size of verifiers concentrates in two values: for $u = 1$, it concentrates in the neighborhood of n^2, and for $u = 2, 3$, it concentrates near $2n^2$.

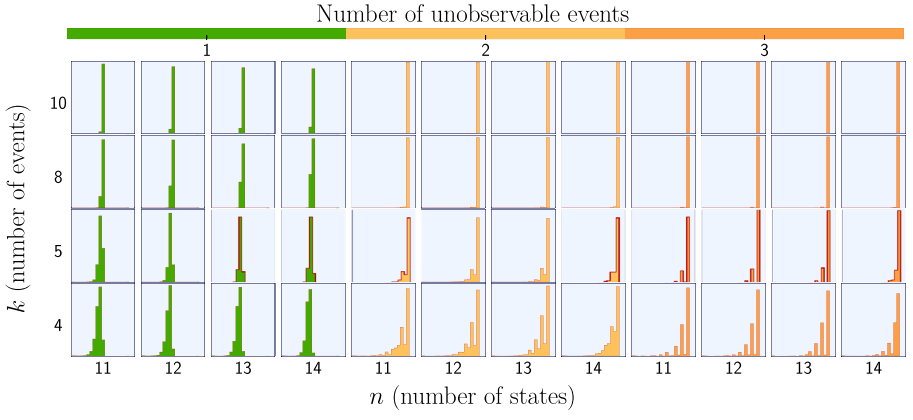
[6] This scale has been chosen since the worst case computational complexity of verifiers proposed by Moreira et al. (2011) and Moreira et al. (2016) is $\mathcal{O}(n^2)$



a



b



c

Fig. 9 Average of observed variables $\hat{\mathbf{V}}^{(n,k,u)}$, $(n,k,u) \in I_1$ (**a**); Standard deviation of variable $\hat{\mathbf{V}}^{(n,k,u)}$, $(n,k,u) \in I_1$ (**b**); Histograms of variables $\hat{\mathbf{V}}^{(n,k,u)}$, $n \in I_{11:14}$, $k \in \{4,5,8,10\}$, $u \in \{1,2,3\}$ (**c**)

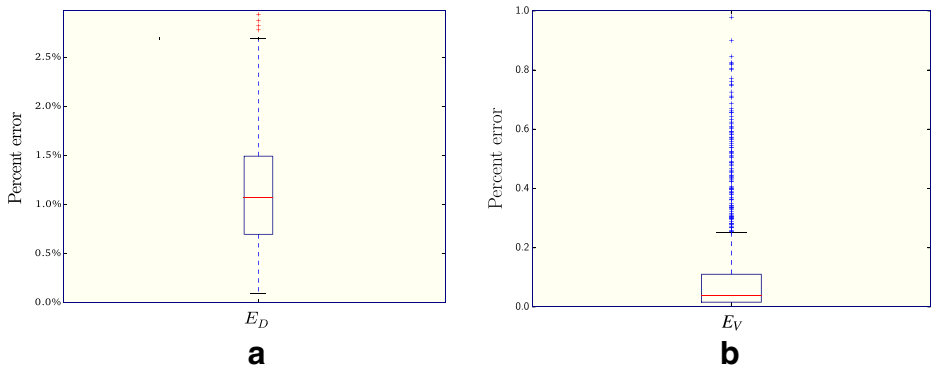


Fig. 10 Box plots of the percent error in the estimation of the sample means $\tilde{\mathbf{D}}^{(n,k,u)}$, $(n,k,u) \in I_1$ (**a**); Box plot of the percent error in the estimation of the mean $\tilde{\mathbf{V}}^{(n,k,u)}$, $(n,k,u) \in I_1$ (**b**)

4.3.3 Checking the correctness of the sample size

We will now check if the sample size (10,000) computed in Section 4.2 has actually made the error in the estimation of the expected value in the construction of diagnosers and verifiers less than 1%. We start by defining the following variables:

- $E_D = \{\rho_D^{(n,k,u)} : (n,k,u) \in I_U\}$: set of points representing the distribution of the percent error in the estimation of the expected values of variables $\mathbf{D}^{(n,k,u)}$ using the central limit theorem. Each value $\rho_D^{(n,k,u)} \in \mathbb{R}$ represents the percent error of the sample mean $\overline{\hat{\mathbf{D}}^{(n,k,u)}}$ when used as an estimate of $\overline{\mathbf{D}^{(n,k,u)}}$, i.e., the expect value of the state size of diagnosers for a set of valid automata $\mathcal{A}_v^{(n,k,u)}$. The value of each $\rho_D^{(n,k,u)}$ was calculated using Eq. 17 with a confidence level equal to 95%.
- $E_V = \{\rho_V^{(n,k,u)} : (n,k,u) \in I_U\}$: set of points that represent the distribution of the percent error in the estimation of the expected values of variables $\mathbf{V}^{(n,k,u)}$ using the central limit theorem. Each value $\rho_V^{(n,k,u)} \in \mathbb{R}$ gives the percent error of the sample mean $\overline{\hat{\mathbf{V}}^{(n,k,u)}}$ as an estimate of $\overline{\mathbf{V}^{(n,k,u)}}$, and was computed using Eq. 17 with a confidence level of 95%.

Figure 10a and b show the box plots[7] of the percent errors in the estimation of the sample means $\tilde{\mathbf{D}}^{(n,k,u)}$ and $\tilde{\mathbf{V}}^{(n,k,u)}$, $(n,k,u) \in I_U$, respectively. From the box plot of ρ_D, shown in Fig. 10a, we can draw the following conclusions regarding the behavior of the percent error in the estimation of the expected values of variables $\mathbf{D}^{(n,k,u)}$, $(n,k,u) \in I_U$, as follows: *(i)* the medians indicate that the error is approximately equal to 1%; *(ii)* the third quartile indicate that typical errors can be as large as 1.5%; *(iii)* the maximum value of the error is approximately 3%. The box plot E_V shown in Fig. 10b allows us to draw the following

[7] A box plot is a graphical display that provides a visual representation of the five-number summary of a data set: first quartile Q1, median, third quartile Q3 and the maximum value. The box of the plot contains the central 50% of the distribution, from the first (Q_1) to the third (Q_3) quartile. A line inside the box marks the median. The lines extending from the box are called whiskers, and encompass the rest of the data, except for potential outliers (data that lie more than $1.5IQR = 1.5(Q_3 - Q_1)$ below the Q_1 and above Q_3), which are shown separately (Hair et al. 2007).

conclusions regarding the percent error in the estimation of the average state size of verifiers: *(i)* the median of the error is approximately 0.03%; *(ii)* the third quartile indicates that typical errors can reach 0.1%; *(iii)* only outliers can reach 1%. We can, therefore, conclude that the choice of a sample size equal to 10,000 provides statistical conditions closed to that specified for diagnosers and verifiers.

4.4 Empirical models for the average state sizes of verifiers and diagnosers

Based on the results of the experiments with uniform sampling, we will now propose experimental models for the average state sizes of verifiers and diagnosers.

4.4.1 An empirical model for the average state size of verifiers

Let $\mathbf{M}_V(n,k,u)$ and $\hat{\mathbf{M}}_V(n,k,u)$ denote the functions that map a point $(n,k,u) \in I_U$ to the expected values of the state size of the verifiers built for the elements of $\mathcal{A}_v^{(n,k,u)}$ (of all valid automata), and to the sample mean of the state sizes of verifiers of the elements of $\hat{\mathcal{A}}_v^{(n,k,u)}$ (of all valid automata generated with uniform distribution), respectively. As shown in Section 4.3.3, with sample size $N = 10,000$, $\hat{\mathbf{M}}_V(n,k,u)$ is a very close estimation of the expected value $\mathbf{M}_V(n,k,u)$ of the state size of verifiers (with around 0.1% of error). Therefore, based on the growth of $\hat{\mathbf{M}}_V(n,k,u)$, we will propose an empirical model for the average state size of verifiers. The strategy adopted here is to carry out nonlinear least-square regression on the contours of $\hat{\mathbf{M}}_V(n,k_0,u_0)$, where k_0 and u_0 are fixed, using the following nonlinear regression model (Fox and Weisberg 2011):

$$\hat{\mathbf{M}}_V(n,k_0,u_0) = f(n;(a,b)) + e_n, \tag{18}$$

where $f(n;(a,b))$ is a model with parameters a and b, and e_n is the n-th residual for $n \in I_{3:20}$. A numerical algorithm for the minimization of the weighted sum of square residuals

$$S(a,b) = \sum_{n \in I_{3:20}} w_n (\hat{\mathbf{M}}_V(n,k_0,u_0) - f(n;(a,b)))^2, \tag{19}$$

where w_n is a parameter that is used to weigh the fit quality in different points, will be used to find the parameters of each contour.

Function $f(n;(a,b))$ in Eq. 18 is defined (depending on the number of unobservable events), as follows:

- For $u_0 = 1$, $f(n;(a,b)) = (n+a)^b$ for the ten contours $\hat{\mathbf{M}}_V(n,k_0,1)$ that correspond to each value of $k_0 \in I_{2:10} \cup \{16\}$.
- For $u_0 = 2$, $f(n;(a,b)) = 2(n+a)^b$ for the nine contours $\hat{\mathbf{M}}_V(n,k_0,2)$ that correspond to each value of $k_0 \in I_{3:10} \cup \{16\}$.

These functions have been chosen since the results shown in Fig. 9 suggests a quadratic model. Then, we cannot expect to find a lower growth model in average state size of verifiers.

Table 5 presents the regression models for the contours $\hat{\mathbf{M}}_V(n,k_0,u_0)$ that were calculated[8] for $w_n = 1$, $n \in I_{3:20}$. For example, for $k_0 = 3$ (second row of Table 5), the following models were obtained: *(i)* $(n-0.62)^{1.93}$ for $u_0 = 1$, which gives an estimation of the asymptotic growth in the average state size equal to $n^{1.93}$; *(ii)* $2(n-0.88)^{1.92}$ for $u_0 = 2$, which

[8]The calculation was carried out using the open source statistic package R (The R foundation 2016).

Table 5 Regression models of the contours $\hat{\mathbf{M}}_V(n, k_0, u_0)$ used to approximate the growth of the average state size of verifiers

	$u_0 = 1$		$u_0 = 2$	
k_0	Model	$\Theta(\cdot)$	Model	$\Theta(\cdot)$
2	$(n + 0.25)^{1.51}$	$n^{1.51}$	–	–
3	$(n - 0.62)^{1.93}$	$n^{1.93}$	$2(n - 0.88)^{1.92}$	$n^{1.92}$
4	$(n - 0.01)^{1.97}$	$n^{1.97}$	$2(n - 0.42)^{1.98}$	$n^{1.98}$
5	$(n + 0.23)^{1.99}$	$n^{1.99}$	$2(n + 0.19)^{1.99}$	$n^{1.99}$
6	$(n + 0.33)^{1.99}$	$n^{1.99}$	$2(n + 0.11)^{2.00}$	$n^{2.00}$
7	$(n + 0.39)^{2.00}$	$n^{2.00}$	$2(n + 0.06)^{2.00}$	$n^{2.00}$
8	$(n + 0.42)^{2.00}$	$n^{2.00}$	$2(n + 0.04)^{2.00}$	$n^{2.00}$
9	$(n + 0.44)^{2.00}$	$n^{2.00}$	$2(n + 0.03)^{2.00}$	$n^{2.00}$
10	$(n + 0.45)^{2.00}$	$n^{2.00}$	$2(n + 0.02)^{2.00}$	$n^{2.00}$
16	$(n + 0.48)^{2.00}$	$n^{2.00}$	$2(n + 0.00)^{2.00}$	$n^{2.00}$

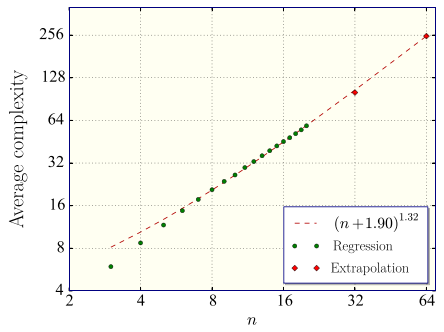
leads to an estimation of the asymptotic growth in the average state size equal to $n^{1.92}$. Figure 11a–f show, using logarithmic scales for both axis, the fit between different contours $\hat{\mathbf{M}}_V(n, k_0, u_0)$ and their corresponding models, represented by the dashed lines in the plots. The values $n = 32$ and $n = 64$ correspond to the experimental results used to validate the model over extrapolated values different from those used in the computation of the regression model. As seen in the plots, there is a significant agreement between each contour $\hat{\mathbf{M}}_V(n, k_0, u_0)$ and the corresponding regression model.

The results obtained in the regression models for the contours of $\hat{\mathbf{M}}_V(n, k_0, u_0)$ indicate that for $k_0 \geq 5$, the average state size of verifiers is quadratic (approximately). Such an observation is illustrated in Fig. 12a and b; Fig. 12a depicts the surface $\hat{\mathbf{M}}_V(n, k, 1)$, $n \in I_{3:20} \cup \{32, 64\}$, $k \in I_{2:10} \cup \{16\}$, i.e., with one unobservable event only, from where, it can be seen that, as expected, it converges to plane $(n+0.48)^2$, and Fig. 12b shows the simultaneous graphical representations of surfaces $\hat{\mathbf{M}}_V(n, k, u)$, $n \in I_{3:20} \cup \{32\}$, $k \in I_{2:10} \cup \{16\}$, $u \in \{2, \ldots, k-1)\}$, i.e., with two or more unobservable events, which are bounded above by plane $2n^2$. As a consequence, the asymptotic growth of the surface $\hat{\mathbf{M}}_V(n, k, u)$, $(n, k, u) \in I_U$ can be empirically modeled as $\Theta(n^2)$. In addition, as observed in Figs. 11 and 12a, b, the empirical model of growth n^2, when extrapolated to values of n and k outside the regression interval, was also verified. Based on these observations, we make the following conjecture.

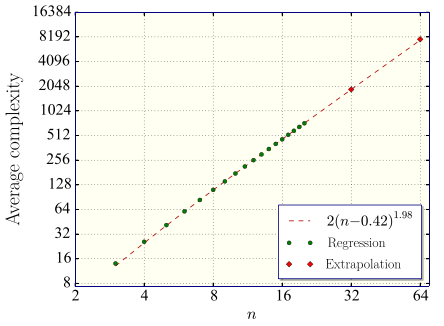
Conjecture 1 The verifiers built in accordance with Moreira et al. (2011) have state size $\Theta(n^2)$, on the average.

4.4.2 An empirical model for the average state size of diagnosers

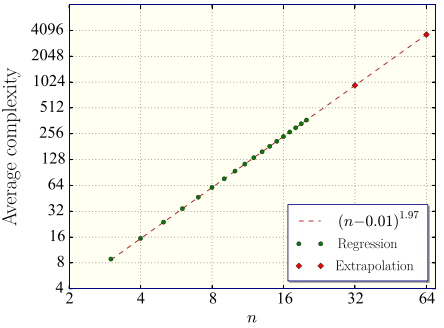
Let $\mathbf{M}_D(n, k, u)$ and $\hat{\mathbf{M}}_D(n, k, u)$ denote the functions that associate a point $(n, k, u) \in I_U$ to the expected value of the computational complexities of the diagnosers built for the elements of set $\mathcal{A}_v^{(n,k,u)}$ of all valid automata, and to the sample means of the state size of the diagnosers built for the set $\hat{\mathcal{A}}_v^{(n,k,u)}$ of all valid automata generated with uniform distribution.



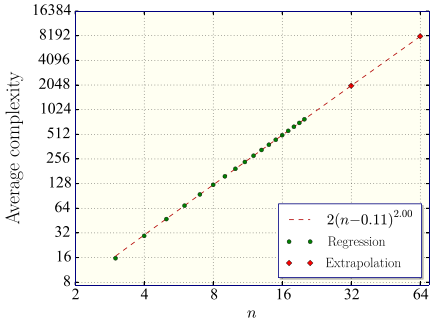
a $\hat{\mathbf{M}}_V(n,3,1)$ and model.



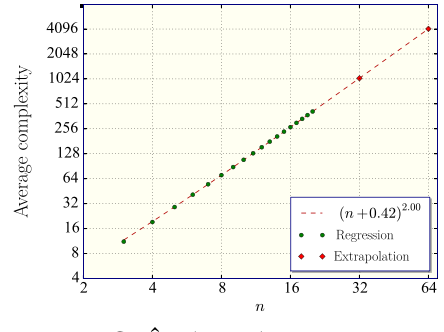
b $\hat{\mathbf{M}}_V(n,4,2)$ and model.



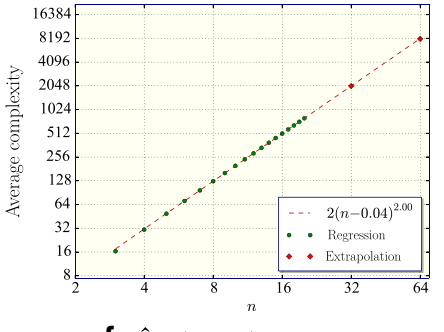
c $\hat{\mathbf{M}}_V(n,4,1)$ and model.



d $\hat{\mathbf{M}}_V(n,6,2)$ and model.



e $\hat{\mathbf{M}}_V(n,8,1)$ and model.



f $\hat{\mathbf{M}}_V(n,8,2)$ and model.

Fig. 11 Plots of the regressions of the contours $\mathbf{M}_V(n,k_0,u_0)$ for a few values of k_0 and u_0. The points in *green* correspond to the values used for the computation of the regression model, the *dashed line* represents the regression model and the *red points* are experimental results used to validate the extrapolation of each model

As shown in Section 4.3.3, the sample size $N = 10,000$ ensures that the sample mean $\hat{\mathbf{M}}_D(n,k,u)$ can estimate the expected value $\mathbf{M}_D(n,k,u)$ with around 1% of percent error. In addition, according to the experimental results of Section 4.3.1, the sample means of the state size of the instances tested in the experiment, are bounded above by

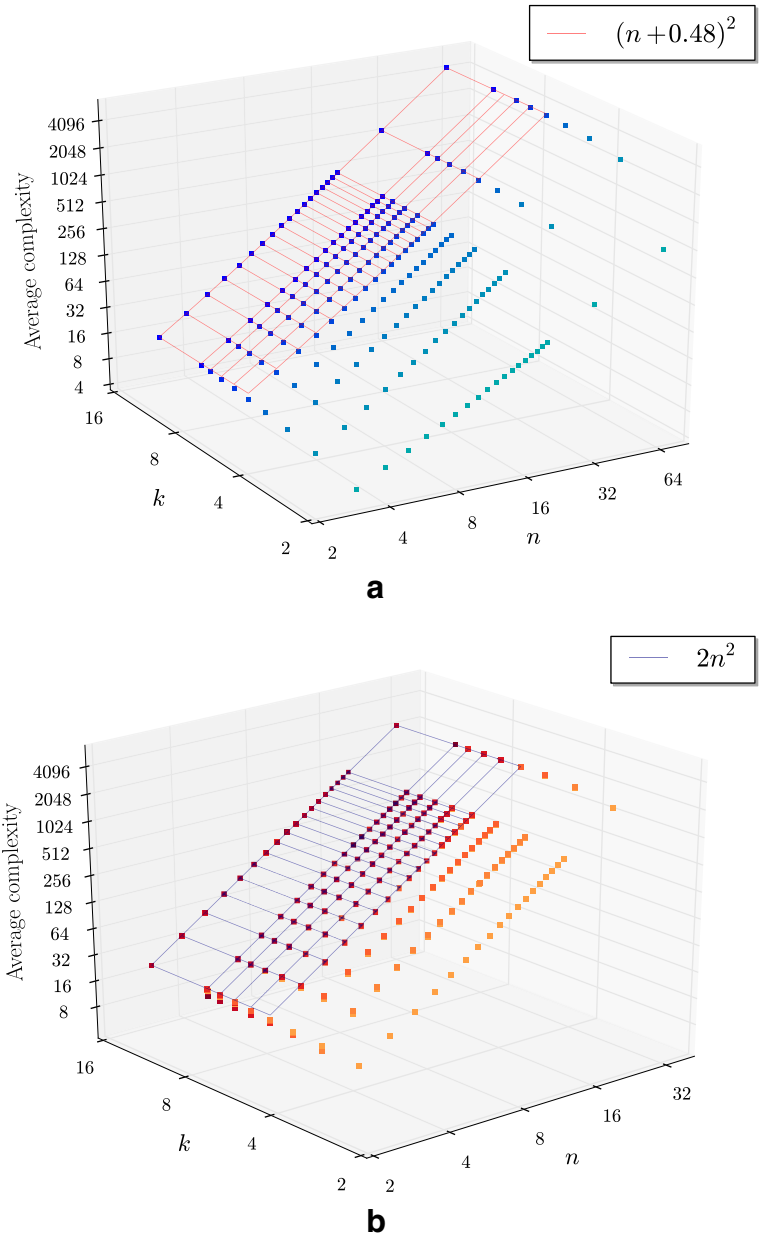


Fig. 12 Convergence of surface $M_v(n,k,1), n \in I_{3:20} \cup \{32,64\}, k \in I_{3:10} \cup \{16\}$ to plane $(n+0.48)^2$ (**a**); convergence of surface $M_v(n,k,u), n \in I_{3:20} \cup \{32\}, k \in I_{3:10} \cup \{16\}, u \in \{2,\ldots,\max(k-1,8)\}$ to plane $2n^2$ (**b**)

the surface $\hat{\mathbf{M}}_D(n,k,1)$, i.e., $\hat{\mathbf{M}}_D(n,k,k-1) \leq \cdots \leq \hat{\mathbf{M}}_D(n,k,2) \leq \hat{\mathbf{M}}_D(n,k,1)$. Therefore, the largest sample mean of the state sizes appears in the sets $\mathcal{A}_v^{(n,k,1)}$ formed with all valid automata with only one unobservable event; Fig. 8a illustrates this fact

Table 6 Empirical models for the estimation of the average state size of diagnosers

k_0	Model	$\Theta(\cdot)$
2	$(n+4.72)^{0.62}$	$n^{0.62}$
3	$(n+1.90)^{1.32}$	$n^{1.32}$
4	$(n+2.29)^{1.63}$	$n^{1.63}$
5	$(n+2.06)^{1.84}$	$n^{1.84}$
6	$(n+1.61)^{2.00}$	$n^{2.00}$
7	$(n+1.36)^{2.13}$	$n^{2.13}$
8	$(n+0.95)^{2.24}$	$n^{2.24}$
9	$(n+0.62)^{2.34}$	$n^{2.34}$
10	$(n+0.30)^{2.42}$	$n^{2.42}$
16	$(n-1.20)^{2.76}$	$n^{2.76}$
24	$(n-2.49)^{3.05}$	$n^{3.05}$

for $\hat{\mathbf{M}}_D(n,k,u)$, $(n,k,u) \in I_1$. This fact suggests that it is possible to use the surface $\hat{\mathbf{M}}_D(n,k,1)$ in order to obtain an empirical model for the average state size of diagnosers.

As in the case of verifiers, in order to estimate the growth of state size of diagnosers, we will use the following non-linear regression model:

$$\hat{\mathbf{M}}_D(n,k_0,1) = (n+a)^b + e_n, \tag{20}$$

where a and b are the parameters of the model, and e_n is the n-th residual for $n \in I_{3:20}$. We use the algebraic model of Eq. 20 since other lower models commonly used (for instance, $\log n$, $n \log n$) are not an upper bound for $\hat{\mathbf{M}}_D(n,k_0,1)$ and, as shown in Fig. 8, we have obtained that the growth of $\hat{\mathbf{M}}_D(n,k_0,1)$ is subexponential. The model parameters will be estimated by minimizing the following weighted sum of the square residuals:

$$S(a,b) = \sum_{n \in I_{3:20}} w_n (\hat{\mathbf{M}}_D(n,k_0,1) - (n+a)^b)^2,$$

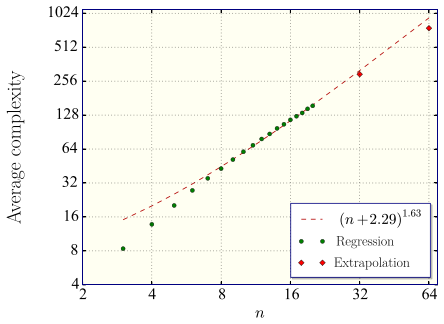
where $w_n = n, n \in I_{3:20}$, since the objective is to identify the asymptotic growth of the curves $\hat{\mathbf{M}}_D(n,k_0,1)$, and, thus, smaller values of n are less important.

Table 6 shows the regression models found for the contours $\hat{\mathbf{M}}_D(n,k_0,1)$, $k_0 \in I_{2,10} \cup \{16,20,24\}$; for example for $k_0 = 6$ (fifth row of the table), the corresponding model is $(n+1.61)^{2.00}$, which leads to $n^{2.00}$, as an estimate for the asymptotic growth.

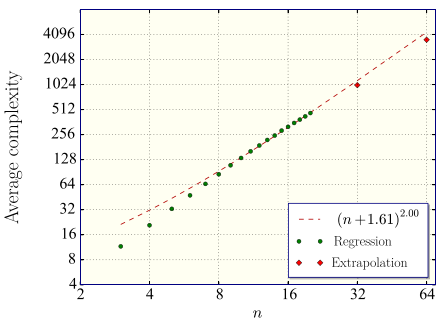
Figure 13a–f show, using a logarithmic scale for both axes, the fit of some of the contours $\hat{\mathbf{M}}_D(n,k_0,u_0)$ of Table 6 together with their corresponding models; each regression model is represented in the graph by a dashed line. As expected, the fit is better for higher values of n, which is due to the choice of weights. Notice that, even the values of the average state sizes for $n = 32$ and $n = 64$ (used for validating the model extrapolation) are bounded above by the regression model.

In order to provide an order of magnitude for the asymptotic growth, we will now investigate the behavior of exponent b in Expression (20) in terms of the number of events k using the following linear model:

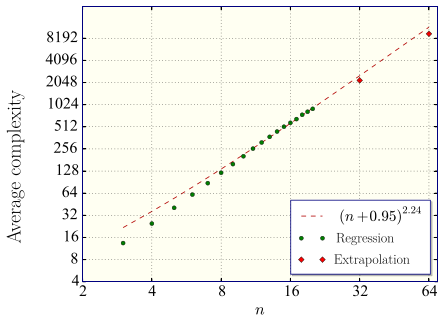
$$b = c_1 \log k_0 + c_2,$$



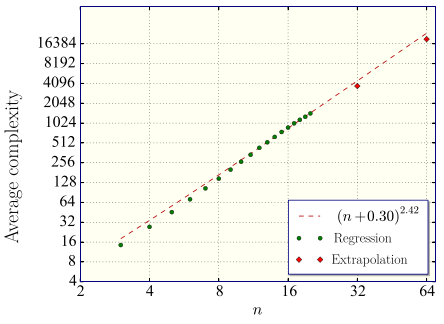
a $\hat{\mathbf{M}}_D(n,4,1)$ and model.



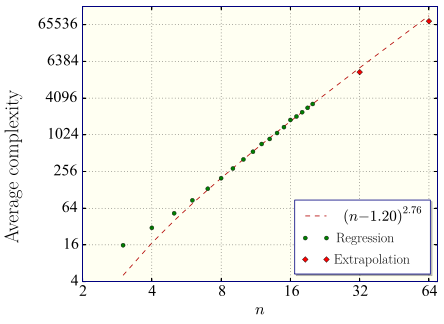
b $\hat{\mathbf{M}}_D(n,6,1)$ and model.



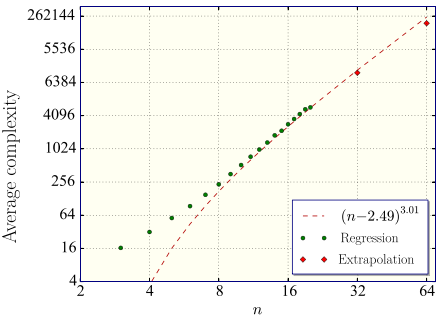
c $\hat{\mathbf{M}}_D(n,8,1)$ and model.



d $\hat{\mathbf{M}}_D(n,10,1)$ and model.



e $\hat{\mathbf{M}}_D(n,16,1)$ and model.



f $\hat{\mathbf{M}}_D(n,24,1)$ and model.

Fig. 13 Plots of the regressions of the contours $\mathbf{M}_D(n,k_0,u_0)$ for a few values of k_0 and u_0. The points in *green* correspond to the values used for the computation of the regression model, the *dashed line* represents the regression model and the *red points* are experimental results used to validate the extrapolation of each model

where c_1 and c_2 are the model parameters and k_0 is the regressor. Carrying out a linear regression based on the values of b of the empirical models of Table 6, we obtain the following linear model:

$$g(n,k) = n^{0.77\log k + 0.63}, \tag{21}$$

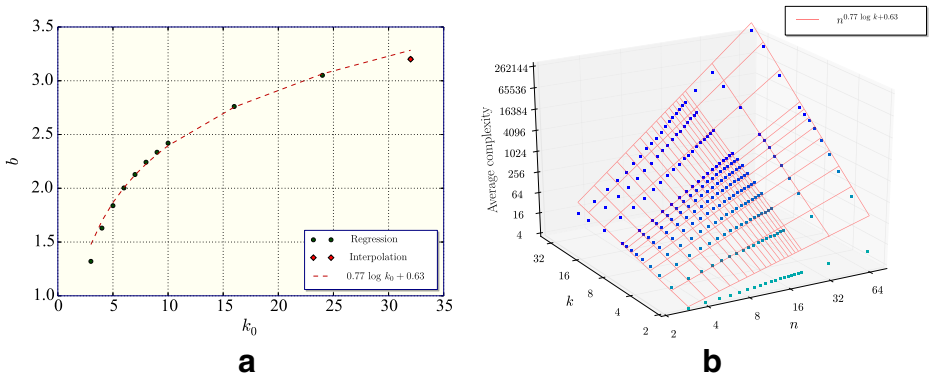


Fig. 14 Curve of exponent b and the regression model as a function of the number of events (**a**); $\mathbf{M}_D(n, k, 1)$ and surface $n^{0.77 \log k+0.63}$ (**b**)

that represents an estimation on the growth of surface $\hat{\mathbf{M}}_D(n, k, 1)$, $n \in I_{3:20} \cup \{32, 64\}$, $k \in I_{2:10} \cup \{16, 24, 32\}$, as shown in Fig. 14b. Therefore, we can make the following conjecture about the average state size of diagnosers.

Conjecture 2 The diagnosers built in accordance with Sampath et al. (1995) have state size $\Theta(n^{0.77 \log k+0.63})$, on the average.

5 Conclusions

The purpose of this paper was to carry out an experimental study on the average state sizes of diagnosers and verifiers used in diagnosability analysis. In this regard, the first contribution of this paper is to introduce a method for experimental analysis of the computational complexity of algorithms proposed in DES theory using deterministic and accessible automata generators (reported in the Computer Science community literature) and data analysis. We also build over the previously known results by proposing an algorithm for exhaustive generation of sets of accessible (not necessarily complete) automata with n states and k events.

By restricting the scope to the specific instances included in this study, the main conclusions that can be drawn from the experimental analysis carried out here are as follows:

1. The diagnoser proposed by Sampath et al. (1995) is, in average, easier to build than expected when we just consider the worst-case computational complexity upper bound given by $O(2^{2n})$. In this regard, it is conjectured here that the state size of diagnosers is $\Theta(n^{0.77 \log k+0.63})$, on the average, where n and k are, respectively, the number of states and events of the input automaton, and was obtained by performing successive regressions of the sample means.
2. The average state size of the verifier proposed by Moreira et al. (2011) is of quadratic order in the number of states of the input automaton. According to the results shown in Sections 4.3.2 and 4.4.1, the state size of verifiers becomes progressively concentrated around n^2 (for automata with one unobservable event and) and $2n^2$ (for automata with more than one unobservable event) with the increase in the number of events of the

input automata. This behavior makes clear that the previously established quadratic worst-case complexity of the algorithm proposed by Moreira et al. (2011) is indeed a tight bound.

It is worth remarking that the algorithm for automaton generation used here does not preclude the possibility of generating non-minimal state automata. How minimal automata may affect the conjectures made here may be a problem of interest for future investigation.

Finally, it is important to remark, as pointed out by McGeoch (2012), that the main objective of studying algorithm performance by means of empirical methods is to get insights about the complexity of algorithms and to make quantitative hypothesis on their complexities that can be either extended or validated by creating a theory that explains the observed behavior.

Acknowledgments The authors are in debit with Prof. Stéphane Lafortune, University of Michigan, Ann Arbor, for the encouragement to submit this paper for publication and for suggesting its title. They also would like to thank the Brazilian Research Council (CNPq) for financial support.

References

Almeida M, Moreira N, Reis R (2007) Enumeration and generation with a string automata representation. Theor Comput Sci 387(2):93–102

Bassino F, David J, Nicaud C (2009) Enumeration and random generation of possibly incomplete deterministic automata. Pure Mathematics and Applications 19(2-3):1–16

Bassino F, Nicaud C (2007) Enumeration and random generation of accessible automata. Theor Comput Sci 381(1-3):86–104

Clavijo LB (2014) Aspectos computacionais associados á implementação de algoritmos para sistemas a eventos discretos. Universidade Federal de Rio de Janeiro, Tese de doutorado

Fox J, Weisberg S (2011) An R companion to applied regression SAGE publications. USA, Thousand Oaks

Gubner JA (2006) Probability and random processes for electrical and computer engineers. Cambridge University Press, New York

Hair J, Anderson R, Tatham R (2007) Análise Multivariada de Dados. Bookman, Porto Alegre

Jiang S, Huang Z, Chandra V, Kumar R (2001) A polynomial algorithm for testing diagnosability of discrete-event systems. IEEE Trans on Automatic Control 46(8):1318–1321

Jirásková G, Masopust T (2012) On a structural property in the state complexity of projected regular languages. Theor Comput Sci 449:93–105

McGeoch C (2012) A guide to experimental algorithmics. Cambridge University Press, New York

Moore F (1971) On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Trans Comput 20(10):1211–1214

Moreira MV, Basilio JC, Cabral FG (2016) "Polynomial time verification of decentralized diagnosability of discrete event systems" vs. "Decentralized failure diagnosis of discrete event systems": a critical appraisal. IEEE Trans Autom Control 61(1):178–181

Moreira MV, Jesus TC, Basilio JC (2011) Polynomial time verification of decentralized diagnosability of discrete event systems. IEEE Trans Autom Control 56(7):1679–1684

Qiu W, Kumar R (2006) Decentralized failure diagnosis of discrete event systems. IEEE Trans on Systems, Man and Cybernetics, Part A 36(2):384–395

Ramadge PJ, Wonham WM (1989) The control of discrete-event systems. Proc of the IEEE 77:81–98

Reis R, Moreira N, Almeida M (2005) On the representation of finite automata. In: Proceedings of the 7th international workshop on descriptional complexity of formal systems, pp 269–276

Sampath M, Sengupta R, Lafortune S, Sinnamohideen K, Teneketzis D (1996) Failure diagnosis using discrete event models. IEEE Trans on Control Systems Technology 4(2):105–124

Sampath M, Sengupta R, Lafortune S, Sinnamohideen K, Teneketziz D (1995) Diagnosability of discrete event systems. IEEE Trans Autom Control 40(9):1555–1574

The R foundation (2016) The R Project for Statistical Computing. Website. https://www.r-project.org/
Wong KC (1998) On the complexity of projections of discrete-event systems. In: In IEE workshop on discrete event systems, pp 201–208
Yoo T-S, Lafortune S (2002) Polynomial-time verification of diagnosability of partially observed discrete-event systems. IEEE Trans Autom Control 47(9):1491–1495

Leonardo Bermeo Clavijo was born on August 21, 1971 in Bogotá, Colombia. He received the Electrical Engineering degree in 1999 from the Distrital University Francisco Jose de Caldas, Bogotá, Colombia, the M.Sc. degree in Industrial Automation from the National University of Colombia, Bogotá, Colombia, in 2004, and the D.Sc. degree in Electrical Engineering from the Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, in 2014. He began his career in 1999 as an Assistant Lecturer at the Department of Computer Systems Engineering of the University of Cundinamarca, Cundinamarca, Colombia. Since 2004, he has been a Full Professor in the Electrical Engineering Department at the National University of Colombia. He served as Academic Chair for the Master Degree Program in Industrial Automation at the same department from July, 2006, to August, 2009. His current interest is in the application of discrete event system theory for modeling clinical procedures in newborns.

João C. Basilio was born on March 15, 1962 in Juiz de Fora, Brazil. He received the Electrical Engineering degree in 1986 from the Federal University of Juiz de Fora, Juiz de Fora, Brazil, the M.Sc. degree in Control from the Military Institute of Engineering, Rio de Janeiro, Brazil, in 1989, and the Ph.D. degree in Control from Oxford University, Oxford, U.K., in 1995. He began his career in 1990 as an Assistant Lecturer at the Department of Electrical Engineering of the Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, and, since 2014, has been a Full Professor in Control at the same department. He served as Academic Chair for the Control and Automation Engineering course of Polytechnic School of the Federal University of Rio de Janeiro from January, 2005, to December, 2006, as Chair for the Electrical Engineering Post-graduation Program (COPPE) from January, 2008, to February, 2009, as Head of the Electrical Engineering Department, from May, 2012 to February, 2014, and since 2014 he has been the Dean of Polytechnic School. From September, 2007, to December, 2008, he spent a sabbatical leave at the University of Michigan, Ann Arbor, and was an Invited Professor of École Centrale of Lille, University of Lille, France, during September, 2016. His current interests are fault diagnosis and supervisory control of discrete-event systems. Prof. Basilio is the recipient of the Correia Lima Medal.