On Algorithms verifying Initial-and-Final-State Opacity: Complexity, Special Cases, and Comparison

Tomáš Masopust and Petr Osička

Faculty of Science, Palacky University Olomouc, Czechia

Abstract

Opacity is a general framework modeling security properties of systems interacting with a passive attacker. Initial-and-final-state opacity (IFO) generalizes the classical notions of opacity, such as current-state opacity and initial-state opacity. In IFO, the secret is whether the system evolved from a given initial state to a given final state or not. There are two algorithms for IFO verification. One arises from a trellis-based state estimator, which builds a semigroup of binary relations generated by the events of the automaton, and the other is based on the reduction to language inclusion. The time complexity of both algorithms is bounded by a super-exponential function, and it is a challenging open problem to find a faster algorithm or to show that no faster algorithm exists. We discuss the lower-bound time complexity for both general and special cases, and use extensive benchmarks to compare the existing algorithms.

Key words: Discrete-event systems, finite automata, initial-and-final-state opacity, verification, complexity, benchmarks

1 Introduction

Opacity is a framework to specify security properties of systems interacting with a passive attacker. The attacker is an observer with a complete knowledge of the structure of the system and with a limited observation of its behavior. The secret is given either as a set of states or as a set of strings. The attacker estimates the behavior of the system based on its observations, and the system is opaque if the attacker never ascertains the secret. For more information, we refer to Jacob et al. (2016).

For automata models, several notions of opacity have been investigated in the literature, including current-state opacity (CSO), initial-state opacity (ISO), and initial-and-final-state opacity (IFO). Whereas CSO prevents the attacker from discovering that the system is currently in a secret state, ISO prevents the attacker from discovering that the system started in a secret state. Wu and Lafortune (2013) introduced IFO as a generalization of both CSO and ISO.

The applicability of theoretical results depends upon the computational efficiency of the developed algorithms. The fundamental question concerns the time complexity of the algorithms. In particular, a tight complexity characterization provides an insight into the size of instances the algorithms are able to handle in an acceptable time and helps us understand the notion of an optimal algorithm. For (super-)exponential algorithms, the question of the existence of faster algorithms is particularly important, and lies in the core of the P vs. NP problem.

For selected notions of opacity, Wintenberg et al. (2022) and Balun et al. (2024) show that the existing algorithms can handle models with a few hundreds or thousands of states, which is far from industrial needs. For IFO, we show that the existing algorithms can handle only very small instances. In Figure 1, we give an example of an automaton with five states and 13 events that our tools implementing current algorithms are not able to solve in 48 hours.

One of the purposes of this paper is to show that the situation is not hopeless as the example suggests. By making use of efficient tools for language-inclusion testing, we are able to verify IFO for significantly larger instances. In particular, we are able to verify the example of Figure 1 in a few milliseconds. To the best of our knowledge, no research on efficiency and optimization of algorithms for opacity verification has been done in the literature so far.

Whereas the worst-case time complexity to verify most of the opacity notions is exponential, and tight under the assumptions discussed below, the time complexity of current algorithms to verify IFO is super-exponential and it is not known whether there is an exponential-time algorithm or whether the super-exponential time complexity is tight. We

⋆ Corresponding author: T. Masopust.
Email addresses: tomas.masopust@upol.cz (Tomáš Masopust), petr.osicka@upol.cz (Petr Osička).

show that for the algorithms discussed so far in the literature, the super-exponential time complexity is tight.

The (non)existence of an exponential-time algorithm remains a challenging open problem. If there were such an algorithm, our results imply that it would require new ideas and verification techniques. To prove the nonexistence of such an algorithm is even more intricate, because no techniques are known regarding how to show that there is no algorithm of a given complexity; consider, for instance, the questions of the separation of complexity classes in complexity theory or of the existence of a polynomial algorithm for prime factorization.

Current algorithms to verify IFO. There are two algorithms verifying IFO in the literature—the trellis-based algorithm of Wu and Lafortune (2013), and the algorithm using a reduction to language inclusion of Balun et al. (2023).

The trellis-based algorithm uses a state estimator of Saboori and Hadjicostis (2013) building a semigroup of binary relations defined by events of a given automaton. The inclusion algorithm is based on the language inclusion of two automata. It computes the product of one automaton with the complement of the other, and checks emptiness. For nondeterministic automata, the involved complementation requires the construction of the observer.

Considering time complexity, the upper bounds on existing algorithms are super-exponential of order $O^*(2^{n^2})$, where n is the number of states of the automaton. The IFO verification problem is PSpace-complete, and the generally accepted assumption that PTime is different from PSpace implies that there is no polynomial-time algorithm to verify IFO.

However, there could be a sub-exponential-time algorithm. To (conditionally) exclude its existence, Impagliazzo and Paturi (2001) formulated a strong exponential time hypothesis motivated by the fact that there is so far no algorithm solving SAT significantly faster than trying all possible truth assignments. The hypothesis states that, for any constant $c < 2$, there is a sufficiently large k such that k-SAT (SAT with each clause of the formula containing no more than k literals) cannot be solved in time $O(c^n)$, where n is the number of variables of the formula. Under this hypothesis, Balun et al. (2023) showed that there is no algorithm to verify IFO of an n-state automaton in time $O^*(2^{n/(2+\varepsilon)})$, for any $\varepsilon > 0$.

The question that remains is whether we can verify IFO in exponential time $O^*(2^n)$ rather then in super-exponential time $O^*(2^{n^2})$. This is a significant difference; compare, for example, $2^5 = 32$ and $2^{5^2} = 33,554,432$.

Our contributions. First, we discuss the time complexity of both existing IFO-verification algorithms and show that their tight time complexity is super-exponential. Then, we show that the trellis-based algorithm is a special case of the other.

Lower bound	Upper bound	Condition
$\Omega(2^{n^2})$	$O^*(2^{n^2})$	none
$O^*(2^n)$	$O^*(2^n)$	$Q_{NS} = I_{NS} \times F_{NS}$
$O^*(2^{n \log n})$	$O^*(2^{n \log n})$	deterministic
poly	poly	observer property
$O^*(2^{n(n+1)/2})$	$O^*(2^{n(n+1)/2})$	self-loops nondet.
$O^*((n+1)!)$	$O^*((n+1)!)$	self-loops det.

Table 1
Upper and lower bounds on the time complexity of existing algorithms for IFO verification. If they coincide, then the bound is tight. Here n stands for the number of states of the automaton.

Second, we discuss several special cases. Namely, we show that (i) for the set of nonsecret pairs in the form of a Cartesian product, the complexity drops to $O^*(2^n)$, (ii) for deterministic automata, the complexity drops to $O^*(2^{n \log n})$, (iii) for automata satisfying the observer property of Wong and Wonham (1996), the complexity is polynomial, and (iv) for automata where all cycles are in the form of self-loops, the complexity drops to $O^*(2^{n(n+1)/2})$, resp. to $O^*((n+1)!)$ for deterministic automata, see Table 1.

Third, we design new algorithms using advanced tools for language-inclusion testing and create extensive benchmarks, based on real data, to compare the existing and our new IFO-verification algorithms. Our results show that the new algorithms perform better and are able to verify larger instances. The algorithms and benchmarks are available at `https://apollo.inf.upol.cz:81/masopust/ifo-benchmarks`.

2 Preliminaries

We assume that the reader is familiar with automata theory, see Hopcroft and Ullman (1979). For a set S, the cardinality of S is denoted by $|S|$ and the power set of S by 2^S. An alphabet Σ is a finite nonempty set of events, partitioned into Σ_o and Σ_{uo}, of the observable and unobservable events, respectively. A string over Σ is a finite sequence of events from Σ. The set of all strings over Σ is denoted by Σ^*, and ε denotes the empty string. A language L over Σ is a subset of Σ^*.

Automata. An *automaton* over an alphabet Σ is a triple $\mathcal{A} = (Q, \Sigma, \delta)$, where Q is a finite set of states and $\delta\colon Q \times \Sigma \to 2^Q$ is a transition function that can be extended to the domain $2^Q \times \Sigma^*$ by induction. The language accepted by $\mathcal{A}$ from a set of states $I \subseteq Q$ by a set of states $F \subseteq Q$ is the set $L_m(\mathcal{A}, I, F) = \{w \in \Sigma^* \mid \delta(I, w) \cap F \neq \emptyset\}$ and the language generated by $\mathcal{A}$ from I is the set $L(\mathcal{A}, I) = L_m(\mathcal{A}, I, Q)$. The automaotn $\mathcal{A}$ is *deterministic* if $|\delta(q, a)| \leq 1$ for every state $q \in Q$ and every event $a \in \Sigma$.

For two sets I and F of states, we use the notation $\mathcal{A}[I, F]$ to denote a copy of $\mathcal{A}$ where I is the set of initial states and F is the set of final states. Notice that $\mathcal{A}[I, F]$ is the classical *nondeterministic finite automaton* (NFA), and that it is a *deterministic finite automaton* (DFA) if it is deterministic

and has a single initial state. To specify the components of $\mathcal{A}[I, F]$, we use the standard notation $\mathcal{A} = (Q, \Sigma, \delta, I, F)$. If a set is a singleton, we simply write its element; for instance, if $I = \{i\}$, we write $\mathcal{A} = (Q, \Sigma, \delta, i, F)$.

For automata $\mathcal{A}_i = (Q_i, \Sigma_i, \delta_i)$, for $i = 1, \ldots, n$ and $n \geq 2$, if the sets Q_i and Q_j are disjoint whenever $i \neq j$, then the *nondeterministic union* of $\mathcal{A}_1, \ldots, \mathcal{A}_n$ is the automaton $\mathcal{A} = (\bigcup_{i=1}^n Q_i, \bigcup_{i=1}^n \Sigma_i, \bigcup_{i=1}^n \delta_i)$ formed by union of components; since every function is a relation by definition, we can view the transition function δ_i as a subset of $Q_i \times \Sigma_i \times Q_i$, which justifies the definition of $\mathcal{A}$. Then, for every subsets I and F of $\bigcup_{i=1}^n Q_i$, we have $L(\mathcal{A}, I) = \bigcup_{i=1}^n L(\mathcal{A}_i, I \cap Q_i)$ and $L_m(\mathcal{A}, I, F) = \bigcup_{i=1}^n L_m(\mathcal{A}_i, I \cap Q_i, F \cap Q_i)$.

The *disjoint union* of automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ first makes the state sets of the automata pairwise disjoint by a suitable renaming of the states, together with the corresponding adjustment of the transition functions, and then performs the nondeterministic union on the resulting automata.

Projections of strings, languages, and automata. A *projection* $P\colon \Sigma^* \to \Sigma_o^*$ is a morphism for concatenation defined by $P(a) = \varepsilon$, for $a \in \Sigma \setminus \Sigma_o$, and $P(a) = a$, for $a \in \Sigma_o$. Intuitively, the action of P is to erase all unobservable events. We lift projections from strings to languages in the usual way.

Let $\mathcal{A}$ be an automaton over Σ, and let $P\colon \Sigma^* \to \Sigma_o^*$ be a projection. The *projected automaton* of $\mathcal{A}$, denoted by $P(\mathcal{A})$, is obtained from $\mathcal{A}$ by replacing every transition (q, a, r) with $(q, P(a), r)$, and by using the classical elimination of ε-transitions. Then $P(\mathcal{A})$ is an automaton over Σ_o, preserving the same observable behavior as $\mathcal{A}$, and with the same set of states as $\mathcal{A}$. The automaton $P(\mathcal{A})$ can be constructed from $\mathcal{A}$ in polynomial time (Hopcroft and Ullman, 1979).

For an NFA $\mathcal{A}$, the reachable part of the DFA constructed from $P(\mathcal{A})$ by the standard subset construction is called the *observer* of $\mathcal{A}$. In the worst case, the observer of $\mathcal{A}$ has exponentially many states compared with $\mathcal{A}$; see Jirásková and Masopust (2012).

IFO. An automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is *initial-and-final-state opaque* (IFO) with respect to sets $Q_S, Q_{NS} \subseteq Q \times Q$ of secret and nonsecret pairs of states, respectively, and a projection $P\colon \Sigma^* \to \Sigma_o^*$, if for every secret pair $(s, t) \in Q_S$ and every string $w \in L_m(\mathcal{A}, s, t)$, there is a nonsecret pair $(s', t') \in Q_{NS}$ and a $w' \in L_m(\mathcal{A}, s', t')$ such that $P(w) = P(w')$.

Asymptotic complexity. Let $g\colon \mathbb{R} \to \mathbb{R}$ be a real function. The class $O(g(n)) = \{f\colon \mathbb{R} \to \mathbb{R} \mid \text{there are } c, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n), \text{ for every } n \geq n_0\}$ consists of functions that do not grow asymptotically faster than g; intuitively, $O(g(n))$ neglects constant factors. Analogously, $O^*(g(n)) = O(g(n)\text{poly}(n))$ neglects constant and polynomial factors. The class $o(g(n)) = \{f\colon \mathbb{R} \to \mathbb{R} \mid \text{for every } c > 0 \text{ there is } n_0 > 0 \text{ such that } |f(n)| < cg(n), \text{ for every } n \geq n_0\}$ consists of functions that grow asymptotically strictly slower than g; that is, $f(n) \in o(g(n))$ if and only if $\lim_{n\to\infty} f(n)/g(n) = 0$. The class of functions that do not grow asymptotically slower than g is denoted by $\Omega(g(n))$ and defined by $f(n) \in \Omega(g(n))$ if and only if $g(n) \in O(f(n))$.

A function $f(n)$ is *super-exponential* if it grows faster than any exponential function of the form c^n, where c is a constant; formally, $\lim_{n\to\infty} f(n)/c^n = \infty$ for all constants $c > 1$.

Semigroup theory and automata. A *semigroup* is a set S together with a binary operation $\cdot$ on S that is associative, i.e., $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.

For an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ with n states, every event $a \in \Sigma$ defines a *binary relation* on Q that can be represented by an $n \times n$ *binary matrix* (a_{ij}), where $a_{ij} = 1$ if $j \in \delta(i, a)$, and $a_{ij} = 0$ otherwise. A *boolean multiplication* of binary matrices (a_{ij}) and (b_{ij}) is defined as the matrix (c_{ij}), where $c_{ij} = \max\{a_{ik}b_{kj} \mid k = 1, \ldots, n\}$, that is, as the classical matrix multiplication where addition is replaced by maximum. The set of all $n \times n$ binary matrices together with the boolean matrix multiplication forms a semigroup $\mathcal{B}_n$ containing 2^{n^2} elements.

Consider the set of binary matrices $\mathcal{G}_\mathcal{A} = \{(a_{ij}) \mid a \in \Sigma\}$ corresponding to the events of $\mathcal{A}$, and denote by $\mathcal{B}_\mathcal{A}$ the semigroup containing all possible finite products of elements of $\mathcal{G}_\mathcal{A}$. Then, $\mathcal{B}_\mathcal{A}$ is a subsemigroup of $\mathcal{B}_n$, and $\mathcal{G}_\mathcal{A}$ is a set of generators of $\mathcal{B}_\mathcal{A}$. In particular, every string $w \in \Sigma^*$ defines a binary matrix $(w_{ij}) \in \mathcal{B}_\mathcal{A}$ representing the relation on Q such that $w_{ij} = 1$ if and only if $j \in \delta(i, w)$.

A fundamental question in semigroup theory is the minimum number of generators of a semigroup. Despite an intensive study of this question for the semigroup $\mathcal{B}_n$ for more than 60 years, the answer is known only for $n \leq 8$, see Hivert et al. (2021) or the sequence A346686 of the On-Line Encyclopedia of Integer Sequences (OEIS). Although the minimum number of generators of $\mathcal{B}_n$ is unknown for $n \geq 9$, Devadze (1968) claimed without proof, and Konieczny (2011) proved, that it grows super-exponentially with respect to n. A lower bound on the number of generators of $\mathcal{B}_n$ is $2^{n^2/4 - O(n)}/(n!)^2$, see (Hivert et al., 2021, Corollary 3.1.8).

Proposition 6 and Theorem 7 in Kim and Roush (1978) further show that (i) for every odd natural number n, there are two binary matrices generating a subsemigroup of $\mathcal{B}_n$ with $((n^2 - 1)/4) \cdot 2^{(n^2-1)/4}$ elements, and (ii) for every integer-valued function f such that $f(n) > n$ and $f(n)/n^2$ tends to zero, the average size of the semigroup generated by two randomly chosen $n \times n$ binary matrices, each with $f(n)$ ones, is at least $2^{(n^2/4)+o(n^2)}$.

Semigroup theory and deterministic automata. Similarly as strings over an automaton correspond to binary relations on its states, the strings over a deterministic automaton $\mathcal{A} = (Q, \Sigma, \delta)$ correspond to partial functions on states.

Algorithm 1. Semigroup-based IFO verification

Input: An automaton $\mathcal{A} = (Q, \Sigma, \delta)$, sets $Q_S, Q_{NS} \subseteq Q \times Q$ of secret and nonsecret pairs, and an alphabet $\Sigma_o \subseteq \Sigma$ of observable events.
Output: `true` iff $\mathcal{A}$ is IFO wrt Q_S, Q_{NS}, and $P\colon \Sigma^* \to \Sigma_o^*$.

1: Construct the projected automaton $P(\mathcal{A})$
2: Compute the elements of the semigroup $\mathcal{B}_{P(\mathcal{A})}$, generated by the events of Σ_o, one by one
3: **for** every newly generated element $(w_{ij}) \in \mathcal{B}_{P(\mathcal{A})}$ **do**
4: **for** every secret pair $(s, f) \in Q_S$ **do**
5: **if** $w_{sf} = 1$ and there is no nonsecret pair
6: $(s', f') \in Q_{NS}$ such that $w_{s'f'} = 1$ **then**
7: **return** `false`
8: **return** `true`

Namely, every $w \in \Sigma^*$ defines a partial function δ_w on Q as follows: for every $q \in Q$, $\delta_w(q) = \delta(q, w)$. Partial functions on Q are called *partial transformations* and, together with the composition of functions, form a so-called *partial-transformation semigroup* denoted by $\mathcal{PT}_n$. The semigroup $\mathcal{PT}_n$ has $(n+1)^n$ elements and is generated by four transformations, see (Howie, 1995, Exercises 12–13, page 41).

The reader can find more information on the size of the subsemigroups of $\mathcal{PT}_n$ generated by less than four transformations in Holzer and König (2004) and Krawetz et al. (2005). For total transformations, we refer to Salomaa (1960) or Dénes (1966).

3 Trellis-Based IFO Verification

In this section, we reformulate the trellis-based algorithm of Wu and Lafortune (2013) in terms of semigroups of binary relations, see Algorithm 1.

Given an automaton $\mathcal{A}$ over Σ, sets Q_S and Q_{NS} of secret and nonsecret pairs of states, and a set of observable events $\Sigma_o \subseteq \Sigma$, the algorithm checks whether every matrix $(w_{ij}) \in \mathcal{B}_{\mathcal{A}}$ generated by the events of $\mathcal{A}$ satisfies the condition: *If there is 1 at a position w_{sf} corresponding to a secret pair $(s, f) \in Q_S$, then there is 1 at a position $w_{s'f'}$ for some nonsecret pair $(s', f') \in Q_{NS}$.*

Considering the worst-case time complexity of Algorithm 1, the construction of the projected automaton $P(\mathcal{A})$ is polynomial in the number of states of $\mathcal{A}$, whereas the computation of the semigroup $\mathcal{B}_{P(\mathcal{A})}$ depends on the structure of $\mathcal{A}$ and may vary significantly. Taking the size of $\mathcal{B}_{P(\mathcal{A})}$ as a parameter, the time complexity of Algorithm 1 is dominated by the cycle on lines 3–7, giving the overall time complexity $O(|\mathcal{B}_{P(\mathcal{A})}| \cdot n^2)$, where n is the number of states of $\mathcal{A}$; indeed, the inner loop on lines 4–7 may be verified by a single scan of all elements of the matrix (w_{ij}).

Before we discuss the maximal size of the semigroup $\mathcal{B}_{P(\mathcal{A})}$, notice that for instances that are IFO, the algorithm has to build the whole semigroup $\mathcal{B}_{P(\mathcal{A})}$, whereas for instances that are not IFO, the algorithm terminates as soon as a matrix that fails the condition is generated.

We now discuss the lower-bound complexity of Algorithm 1.

Theorem 1. *For every $n \geq 1$, there is an automaton $\mathcal{A}_n$ with n states and 2^{n^2} events, all of which are observable, for which Algorithm 1 needs at least 2^{n^2} steps.*

Proof. For every $n \geq 1$, the automaton $\mathcal{A}_n = (Q_n, \Sigma_n, \delta_n)$ is constructed from the semigroup $\mathcal{B}_n$ by taking, for every matrix (a_{ij}) of $\mathcal{B}_n$, an event a that connects state i to state j if and only if $a_{ij} = 1$. Formally, $Q_n = \{1, 2, \ldots, n\}$, $\Sigma_n = \{a \mid (a_{ij}) \in \mathcal{B}_n\}$, and for $i, j \in Q_n$ and $a \in \Sigma_n$, we define $j \in \delta_n(i, a)$ if and only if $a_{ij} = 1$. Then, the semigroup $\mathcal{B}_{\mathcal{A}_n}$ coincides with the semigroup $\mathcal{B}_n$. Consequently, if the IFO instance for the automaton $\mathcal{A}_n$ is positive, then Algorithm 1 has to verify all matrices of $\mathcal{B}_n$, of which there are 2^{n^2}. □

From the construction, we may observe that Algorithm 1 has to make at least 2^{n^2} steps even if we define an event for every generator of $\mathcal{B}_n$ rather than for every element of $\mathcal{B}_n$.

For an illustration, consider the semigroup $\mathcal{B}_2$ with its three generators $(a_{ij}) = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $(b_{ij}) = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$, and $(c_{ij}) = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$. Then, the automaton $\mathcal{A}_2$ constructed in Theorem 1 contains two states, $Q_2 = \{1, 2\}$, three events, $\Sigma_2 = \{a, b, c\}$, and the transition function δ_2 is obtained from the matrices as follows: the matrix (a_{ij}) results in the transitions $\delta(1, a) = \{2\}$ and $\delta(2, a) = \{1\}$, the matrix (b_{ij}) results in the transitions $\delta(1, b) = \{1\}$ and $\delta(2, b) = \{1, 2\}$, and the matrix (c_{ij}) results in the transition $\delta(1, c) = \{1\}$. Since the matrices (a_{ij}), (b_{ij}), (c_{ij}) are generators of the semigroup $\mathcal{B}_2$, each of the 16 elements of $\mathcal{B}_2$ can be written as a multiplications of (some of) the generators. For example, the matrix $\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right) = (a_{ij})(b_{ij})(a_{ij})(b_{ij})$. This matrix corresponds to the relation on states of $\mathcal{A}_2$ defined by the string $abab$; indeed, we have $\delta(i, abab) = \{1, 2\}$, for $i \in \{1, 2\}$. Consequently, $\mathcal{B}_{\mathcal{A}_2} = \mathcal{B}_2$, and if $\mathcal{A}_2$ satisfies IFO, then Algorithm 1 generates all elements of $\mathcal{B}_2$ and requires thus at least 16 steps to verify each of the 16 elements of $\mathcal{B}_2$.

Analogously, considering $\mathcal{B}_5$ with its 13 generators results in the automaton $\mathcal{A}_5$ with five states and 13 events depicted in Figure 1. This automaton is of particular interest, because a tool implementing Algorithm 1 with the automaton $\mathcal{A}_5$, sets $Q_S = Q_{NS} = \{(1, 1), (2, 2), (3, 3), (4, 4), (5, 5)\}$, and the alphabet $\Sigma_o = \Sigma$ as input does not terminate in 48 hours. [1]

Recall that every semigroup $\mathcal{B}_n$ has a minimum number of generators, which are known for $n \leq 8$. For larger n, neither the generators nor their minimum number is known. However, the number is at least $2^{n^2/4 - O(n)}/(n!)^2$, which improves

[1] The choice of $Q_S = Q_{NS}$ only ensures that $\mathcal{A}_5$ satisfies IFO. The same behavior would be observed for any other choice of the sets Q_S and Q_{NS} for which the automaton $\mathcal{A}_5$ satisfies IFO.

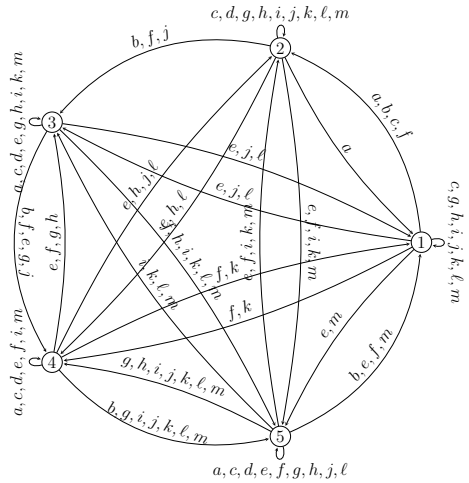


Figure 1. The automaton $\mathcal{A}_5$ on which Algorithm 1 makes at least 2^{25} steps and fails to terminate in 48 hours.

Theorem 1 by decreasing the number of events of the automaton $\mathcal{A}_n$ from 2^{n^2} to an unknown number lower-bounded by the super-exponential function $2^{n^2/4-O(n)}/(n!)^2$.

The reader may wonder whether the complexity of Algorithm 1 drops if the number of events grows asymptotically slower in the number of states than super-exponentially. Unfortunately, Proposition 6 of Kim and Roush (1978) implies that the complexity remains super-exponential even if we consider only automata with two observable events.

Corollary 2. *For every odd $n \geq 1$, there is an automaton $\mathcal{A}_n$ with n states and two observable events, for which Algorithm 1 runs in time $\Omega(n^2 2^{(n^2-1)/4})$.* □

The results so far consider the worst-case time complexity. From the practical point of view, it is of interest to consider the average time complexity of Algorithm 1. As an immediate consequence of Theorem 7 of Kim and Roush (1978), we obtain the following result.

Theorem 3. *Let f be an integer-valued function satisfying $f(n) > n$ and $\lim_{n\to\infty} f(n)/n^2 = 0$. Consider the set $\mathbb{A}_n$ of all automata with the state set $\{1, \ldots, n\}$ and two observable events where either event appears on $f(n)$ transitions. Then, the average runtime of Algorithm 1 on an automaton chosen uniformly at random from $\mathbb{A}_n$ is $\Omega(2^{n^2/4+o(n^2)})$.* □

We further discuss experimental results on an extensive number of data in Section 6. (Results of experiments on randomly generated data are available in the git repository.)

4 Inclusion-Based IFO Verification

The algorithm of Balun et al. (2023) reduces the IFO verification to language inclusion: given two NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$,

Algorithm 2. Classical observer-based IFO verification

Input: An automaton $\mathcal{A} = (Q, \Sigma, \delta)$, sets $Q_S, Q_{NS} \subseteq Q \times 2^Q$ of secret and nonsecret pairs, and the alphabet $\Sigma_o \subseteq \Sigma$ of observable events.
Output: `true` iff $\mathcal{A}$ is IFO wrt Q_S, Q_{NS}, and $P\colon \Sigma^* \to \Sigma_o^*$.
1: Let $\mathcal{A}_S$ be disjoint union of $\mathcal{A}[s,T]$, for $(s,T) \in Q_S$.
2: Let $\mathcal{A}_{NS}$ be disjoint union of $\mathcal{A}[s,T]$, for $(s,T) \in Q_{NS}$.
3: Compute the observer $\mathcal{A}_{NS}^{obs}$ of $\mathcal{A}_{NS}$
4: **return** $L_m(\mathcal{A}_S) \times L_m(\mathcal{A}_{NS}^{obs}) = \emptyset$?

is $L_m(\mathcal{A}_1) \subseteq L_m(\mathcal{A}_2)$? Although there are different optimization techniques discussed in the literature, all existing algorithms for language inclusion are based on the classical observer (aka subset or powerset) construction.

For an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ and a relation $S \subseteq Q \times Q$, we denote by S' the relation over $Q \times 2^Q$ defined by

$$S' = \{(s,T) \mid f \in T \text{ if and only if } (s,f) \in S\} \tag{1}$$

that clusters the elements of S with respect to the domain. To reduce IFO to language inclusion, let $Q_S, Q_{NS} \subseteq Q \times Q$ be the sets of secret and nonsecret pairs, respectively. These sets can be clustered as shown in (1), resulting in the sets Q'_S and Q'_{NS}. Based on these sets, we define the languages

$$\begin{aligned} L_S &:= \bigcup_{(s,T)\in Q'_S} L_m(\mathcal{A}, s, T) \quad \text{and} \\ L_{NS} &:= \bigcup_{(s,T)\in Q'_{NS}} L_m(\mathcal{A}, s, T)\,. \end{aligned} \tag{2}$$

The verification of IFO now reduces to the verification of the inclusion $P(L_S) \subseteq P(L_{NS})$, see Balun et al. (2023).

In the sequel, we distinguish two cases of the verification of language inclusion: (i) the classical approach based on the observer construction, and (ii) approaches reducing the state space of the constructed observer.

Observer-based IFO verification. The languages L_S and L_{NS} of (2) can be represented by NFAs $\mathcal{A}_S$ and $\mathcal{A}_{NS}$, respectively, consisting of no more than n copies of the automaton $\mathcal{A}$, with possibly different initial and final states; see Algorithm 2. As a result, the NFAs $\mathcal{A}_S$ and $\mathcal{A}_{NS}$ have no more than n^2 states each, and the classical verification of the language inclusion $P(L_S) \subseteq P(L_{NS})$ based on the observer construction is of time complexity $O(n^2 2^{n^2}) = O^*(2^{n^2})$.

Note that the upper bound on the time complexity of Algorithm 2 coincides with the upper bound on the time complexity of Algorithm 1. It is not a coincidence. We show that Algorithm 1 is a special case of Algorithm 2.

Lemma 4. *For every automaton $\mathcal{A} = (Q, \Sigma, \delta)$, an alphabet $\Sigma_o \subseteq \Sigma$ of observable events, and sets $Q_S, Q_{NS} \subseteq Q \times 2^Q$ of secret and nonsecret pairs of states, respectively, there is*

an automaton $\mathcal{A}'$ and a bijection between the elements of the semigroup $\mathcal{B}_{P(\mathcal{A})}$ and the states of the observer of $\mathcal{A}'$.

Proof. Let $P\colon \Sigma^* \to \Sigma_o^*$ be the projection erasing unobservable events, and let $\mathcal{G}_{\mathcal{A}} = \{(a_{ij}) \mid a \in \Sigma_o$, and $a_{ij} = 1$ if and only if $j \in \delta(i,a)\}$ be the set of binary matrices corresponding to the events of Σ_o. Then, $\mathcal{G}_{\mathcal{A}}$ is a set of generators of $\mathcal{B}_{P(\mathcal{A})}$, and hence every matrix $(m_{ij}) \in \mathcal{B}_{P(\mathcal{A})}$ is a product of some of the matrices of $\mathcal{G}_{\mathcal{A}}$; say $(m_{ij}) = (a_1)(a_2)\cdots(a_k)$. Let $m = a_1a_2\cdots a_k$ be the corresponding string over Σ_o. We denote the ith row of the matrix (m_{ij}) by (m_{i*}). For $i \in Q$, let $\mathcal{A}_i$ denote a copy of $\mathcal{A}$ with i being the single initial state. Then, the positions of ones in (m_{i*}) correspond to the states of $\mathcal{A}_i$ reachable from the state i under the strings from $P^{-1}(m)$, which corresponds to the state of the observer of $\mathcal{A}_i$ reachable from the state $\{i\}$ under the string m.

We define the automaton $\mathcal{A}'$ as a disjoint union of automata $\mathcal{A}_i$, for all $i \in Q$, and denote by I' the initial states of $\mathcal{A}'$ formed by the initial states of individual $\mathcal{A}_i$'s as renamed by the operation of disjoint union. Then, the positions of ones in (m_{ij}) correspond to the state of the observer of $\mathcal{A}'$ reachable from I' under the string m.

On the other hand, the state of the observer of $\mathcal{A}'$ reachable from the initial state I' under a string m corresponds to the ones in the matrix (m_{ij}).

Thus, there is a one-to-one correspondence between the elements of $\mathcal{B}_{P(\mathcal{A})}$ and the states of the observer of $\mathcal{A}'$. □

As a consequence of Lemma 4, we obtain for Algorithm 2 the same results as for Algorithm 1. Namely, for every $n \geq 1$, there is an automaton $\mathcal{A}_n$ with n states and super-exponentially many events with respect to n such that the observer of the NFA $\mathcal{A}_{NS}$ of Algorithm 2 has 2^{n^2} states; hence, Algorithm 2 has to make at least 2^{n^2} steps. Further, the worst-case time complexity of Algorithm 2 is $\Omega(n^2 2^{(n^2-1)/4})$ even if the automata are over a binary alphabet, and its average time complexity is super-exponential in the number of states of the automaton if the number of transitions grows with the number of states. Similarly to the tool implementing Algorithm 1, our tool implementing Algorithm 2 fails to terminate on the automaton of Figure 1 in 48 hours.

Inclusion-based IFO verification. The reduction to language inclusion is of independent interest, and we formulate it as Algorithm 3. The inclusion problem has been widely investigated in the literature, resulting in many techniques and tools. Algorithm 3 is thus a class of algorithms using different tools on line 3. We consider the state-of-the-art tools in Section 6. For an illustration, these tools require less than a second to verify whether the five-state automaton of Figure 1 is IFO; recall that the tools implementing Algorithms 1 and 2 failed to terminate in 48 hours.

Algorithm 3. General inclusion-based IFO verification

Input: An automaton $\mathcal{A} = (Q, \Sigma, \delta)$, sets $Q_S, Q_{NS} \subseteq Q \times 2^Q$, and $\Sigma_o \subseteq \Sigma$.
Output: `true` iff $\mathcal{A}$ is IFO wrt Q_S, Q_{NS}, and $P\colon \Sigma^* \to \Sigma_o^*$.
1: Let $\mathcal{A}_S$ be disjoint union of $\mathcal{A}[s,T]$, for $(s,T) \in Q_S$.
2: Let $\mathcal{A}_{NS}$ be disjoint union of $\mathcal{A}[s,T]$, for $(s,T) \in Q_{NS}$.
3: **return** $L_m(\mathcal{A}_S) \subseteq L_m(\mathcal{A}_{NS}^{obs})$?

The main idea of the tools is to cut the state space of the constructed observer by keeping only selected states. Imagine, for instance, two sets of states $X \subseteq Y$ computed by the observer. We can keep only Y with the justification that whatever can be computed from X can also be computed from Y.

Although the language-inclusion tools are quite efficient in practice, see Section 6, they are still based on the observer construction. Consequently, the upper bound on the time complexity of Algorithm 3 coincides with that of Algorithms 1 and 2. However, compared with Algorithms 1 and 2, it is an open problem whether this time complexity is also tight for Algorithm 3.

5 Special Cases

In this section, we discuss the worst-case time complexity of Algorithms 1 and 2 for several special cases.

Nonsecret pairs in the form of a Cartesian product. As an immediate consequence of Algorithm 2, we obtain that if the set Q_{NS} is a Cartesian product of states, i.e., $Q_{NS} = I \times F$, then the NFA for L_{NS} coincides with the input automaton $\mathcal{A}$ where the states of I are initial and the states of F are final, i.e., with $\mathcal{A}[I,F]$. In particular, the NFA recognizing the language $P(L_{NS})$ has n states, and hence the inclusion $P(L_S) \subseteq P(L_{NS})$ can be tested in time $O(n^2 2^n) = O^*(2^n)$. Thus, we have the following result of Balun et al. (2023), which improves a similar result of Wu and Lafortune (2013).

Corollary 5. *The IFO property of an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ with respect to secret pairs $Q_S \subseteq Q \times Q$, nonsecret pairs $Q_{NS} = I \times F \subseteq Q \times Q$, and a projection $P\colon \Sigma^* \to \Sigma_o^*$ can be verified in time $O(n^2 2^n)$.* □

Deterministic automata. Another special case arises for deterministic automata with full observation. Given a deterministic automaton $\mathcal{A} = (Q, \Sigma, \delta)$ and a set $S \subseteq Q \times 2^Q$, we first study the number of states of the minimal DFA recognizing the language $\bigcup_{(s,T)\in S} L_m(\mathcal{A}, s, T)$.

Lemma 6. *For a deterministic automaton $\mathcal{A}$ with an n-element state set Q and a set $S \subseteq Q \times 2^Q$, the minimal DFA recognizing the language $\bigcup_{(s,T)\in S} L_m(\mathcal{A}, s, T)$ has no more than $(n+1)^n$ states.*

Proof. The language $\bigcup_{(s,T)\in S} L_m(\mathcal{A}, s, T)$ can be represented by an NFA $\mathcal{A}'$ constructed as a disjoint union of

$\mathcal{A}[s,T]$, for $(s,T) \in S$. Without loss of generality, we assume that $|S| \leq |Q|$; if there were two pairs $(s,T_1),(s,T_2) \in S$, we could replace them by a single pair $(s,T_1 \cup T_2)$. Thus, $\mathcal{A}'$ consists of $|S|$ deterministic components, and hence the state set of the observer of $\mathcal{A}'$ consists of $|S|$-tuples, where on each position, we have either the state of that component, or the information that the component is no longer active. Therefore, the observer of $\mathcal{A}'$ has at most $(n+1)^n$ states. Since the minimal DFA that is language equivalent to $\mathcal{A}'$ does not have more states than the observer of $\mathcal{A}'$, the proof is complete. □

Analogously as the semigroup $\mathcal{B}_n$ is related to automata, is the partial-transformation semigroup $\mathcal{PT}_n$ related to deterministic automata without unobservable events. Therefore, replacing the semigroup $\mathcal{B}_n$ in Algorithm 1 by the semigroup $\mathcal{PT}_n$ implies that the tight worst-case time complexity of Algorithm 1 for deterministic automata with full observation is $O((n+1)^n)$.

Similarly, we immediately have the following theorem giving us the worst-case time complexity of Algorithm 2.

Theorem 7. *If $\mathcal{A}$ is an n-state deterministic automaton without unobservable events, then the time complexity of Algorithm 2 is bounded from above by $O(n^2(n+1)^n)$ and from below by $(n+1)^n$.*

Proof. Consider the languages L_S and L_{NS} as defined in (2). Lemma 6 shows that the number of states of the observer of the automaton for the language L_{NS} is at most $(n+1)^n$. Since the number of states of the automaton for L_S is at most n^2, the time complexity of Algorithm 2 is $O(n^2(n+1)^n)$.

For the other part, we consider the semigroup $\mathcal{PT}_n$ generated by four transformations, say t_1,t_2,t_3,t_4. We construct a deterministic automaton $\mathcal{A}$ with n states $Q=\{1,\ldots,n\}$ by defining the alphabet $\Sigma=\{t_1,t_2,t_3,t_4\}$ and the transition function $\delta(q,t_i)=t_i(q)$, for every $q \in Q$ and every $t_i \in \{t_1,t_2,t_3,t_4\}$. Since every transformation $t \in \mathcal{PT}_n$ is a composition of the four transformations t_1,t_2,t_3,t_4, we can see t as a string w_t over Σ, and hence $\delta(q,w_t)=t(q)$, for every $q \in Q$. Then, for $Q_{NS}=\{(i,i) \mid i \in Q\}$, the automaton $\mathcal{A}_{NS}$ of Algorithm 2 is a disjoint union of automata $\mathcal{A}_1,\ldots,\mathcal{A}_n$, where $\mathcal{A}_i=\mathcal{A}[i,i]$ is a copy of $\mathcal{A}$ with state i initial and final. Then, the observer of $\mathcal{A}_{NS}$ has as many states as there are transformations in $\mathcal{PT}_n$, which is $(n+1)^n$, and hence Algorithm 2 makes at least that many steps. □

Observer property. The *observer property* was introduced by Wong and Wonham (1996) and, as pointed out by Feng and Wonham (2010), is equivalent to the *observation equivalence* in Hennessy and Milner (1980). Given a DFA $\mathcal{A}$ over Σ generating the language L and accepting the language L_m, and a set of observable events $\Sigma_o \subseteq \Sigma$, the projection $P\colon \Sigma^* \to \Sigma_o^*$ is an *L_m-observer* if for all strings $t \in P(L_m)$ and $s \in L$, whenever the string $P(s)$ is a prefix of t, then there exists a string $u \in \Sigma^*$ such that $su \in L_m$ and $P(su)=t$.

Wong (1998) has shown that, under the observer property, the observer of $\mathcal{A}$ does not have more states than the automaton $\mathcal{A}$ itself, and that the observer of $\mathcal{A}$ may be computed in polynomial time with respect to the size of $\mathcal{A}$; see also Wong and Wonham (2004) and Feng and Wonham (2010). In combination with Algorithm 2, we have the following.

Theorem 8. *Given a deterministic automaton $\mathcal{A}$ over Σ, the sets Q_S and Q_{NS} of secret and nonsecret pairs, and a set of observable events $\Sigma_o \subseteq \Sigma$, let $\mathcal{A}_{NS}$ be the disjoint union of automata $\mathcal{A}[s,T]$, for $(s,T) \in Q_{NS}$. If the projection $P\colon \Sigma^* \to \Sigma_o^*$ is an $L_m(\mathcal{A}_{NS})$-observer, then the time to verify IFO is polynomial.* □

Remark 9. Although the definition of the observer property is for DFAs, it can be applied to NFAs. Indeed, every nondeterministic choice (p,a,q) and (p,a,r), with $q \neq r$, can be replaced by three transitions (p,u,p'), (p',a,q), and (p,a,r), where p' is a new state and u is a new unobservable event. The construction results in a DFA, the observer of which is isomorphic to the observer of the original automaton.

Partially ordered automata. Partially ordered automata, aka 1-weak, very weak, linear, acyclic, or extensive automata, see, e.g., Masopust and Krötzsch (2021), are automata where the transition relation induces a partial order on states. This restriction implies that as soon as a state is left during the computation, it is never visited again. Said differently, the only cycles in the automaton are self-loops. Let $\mathcal{A}=(Q,\Sigma,\delta)$ be an automaton. The reachability relation $\leq$ on states is defined by setting $p \leq q$ if there is a string $w \in \Sigma^*$ such that $q \in \delta(p,w)$. The automaton $\mathcal{A}$ is *partially ordered* if the reachability relation $\leq$ is a partial order.

Partially ordered automata recognize a subclass of regular languages strictly included in *star-free languages*, see Brzozowski and Fich (1980) and Krötzsch et al. (2017). Star-free languages are languages definable by *linear temporal logic*, which is a logic widely used as a specification language in automated verification.

When we order the states of a partially ordered automaton by the reachability relation $\leq$, the matrices of binary relations corresponding to strings over the partially ordered automaton are upper triangular. Upper triangular binary matrices form a semigroup, denoted by $\mathcal{UT}_n$, that has $2^{n(n+1)/2}$ elements and a (unique) minimal generating set with $n(n+1)/2+1$ elements; one of the generators is the identity matrix, see Hivert et al. (2021). Therefore, for every $n \geq 1$, there is a partially ordered automaton $\mathcal{A}_n$ with n states and $n(n+1)/2$ events (we do not need an event corresponding to the identity matrix) such that (i) the semigroup $\mathcal{B}_{P(\mathcal{A}_n)}$ constructed in Algorithm 1 has $2^{n(n+1)/2}$ elements, and (ii) the observer constructed in Algorithm 2 has $2^{n(n+1)/2}$ states. Consequently,

	Negative instances		Positive instances	
Tool	1 min.	5 min.	1 min.	5 min.
vata	769	583	671	518
trellis	120	64	867	720
faudes	485	71	402	58
mata	467	55	397	49
limi	47	1	56	4

Table 2
The numbers of instances not solved within a given time.

we have the following super-exponential worst-case time complexity of the algorithms.

Theorem 10. *The worst-case time complexity of Algorithms 1 and 2 for partially ordered automata is* $O^*(2^{n(n+1)/2})$*, and this result is tight.* □

Similarly, we could discuss deterministic partially ordered automata, the strings over which correspond to binary upper-triangular matrices with up to one nonzero element in each row. The semigroup of such matrices has $(n+1)!$ elements and $n(n-1)/2$ generators, resulting in the tight worst-case time complexity $O^*((n+1)!)$ of Algorithms 1 and 2 for deterministic partially ordered automata. Note that deterministic partially ordered automata are weaker than their nondeterministic counterpart, see Krötzsch et al. (2017).

6 Experimental Comparison – Benchmarks

We implemented Algorithm 1 in the *trellis* tool and Algorithm 2 in the *faudes* tool. We call the latter tool *faudes* because it uses the core of the C++ library `libFAUDES`.[2] Algorithm 3 is a schema implemented as a transducer taking an IFO instance and creating a language-inclusion instance that is subsequently verified by an external tool. Included tools are *vata* of Abdulla et al. (2010), *limi* of Černý et al. (2017), and *mata* of Chocholatý et al. (2024).[3]

We ran experiments on an Ubuntu 22.04.3 LTS machine with 32 Intel(R) Xeon(R) CPU E5-2660 v2@2.20GHz processors and 246 GB memory, executing 30 experiments in parallel, using the `parallel` tool of Tange (2023). Each tool was given a timeout of five minutes. The data and tools are available at `https://apollo.inf.upol.cz:81/masopust/ifo-benchmarks`, where we further present results of experiments on random data.

Table 2 summarizes the number of instances the tool did not solve within the given timeout. The best performance is obtained by *limi*, which solved almost all instances within five minutes. It is worth noticing that *trellis* performs very well for negative instances, but very badly for positive instances. This observation is in accordance with theoretical results where, for negative instances, the computation of *trellis* stops as soon as a counterexample is found, whereas for positive instances, each element of the semigroup has to be verified.

Tool	Negative instances	Positive instances
vata	51.233 / 298.907 / 0.004	58.317 / 289.707 / 0.005
trellis	14.985 / 298.882 / 0.003	91.045 / 294.742 / 0.004
faudes	102.447 / 295.533 / 0.006	82.658 / 297.417 / 0.006
mata	81.044 / 299.024 / 0.006	60.948 / 291.438 / 0.005
limi	20.555 / 135.465 / 0.188	18.652 / 291.571 / 0.205

Table 3
Average/maximum/minimum computation times in seconds.

The average, maximum, and minimum computation times of the tools are summarized in Table 3. On average, *limi* performs the best on both negative and positive instances, while *trellis* performs the best for negative instances. The reason why the performance of *trellis* is bad for positive instances was discussed above. Its success for negative instances, on the other hand, may come from the possibility of quickly arriving at a counterexample if IFO is not satisfied. However, the question why *limi* performs so well on both types of instances is an open problem that may suggest the existence of a theoretically faster algorithm for IFO verification.

Figure 2 shows the number of instances solved by the tools in five minutes. It confirms that *limi* is the best tool to verify IFO, perhaps run with *trellis* in parallel to quickly catch negative instances. It is a challenging question whether the combination of the algorithms behind these tools may result in a better algorithm or help us answer the open problem of the complexity of the IFO-verification problem.

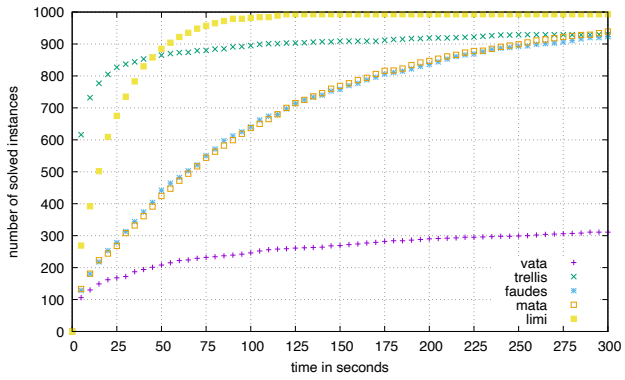
Figure 3 shows the time to solve instances of particular sizes. Except for the case of *trellis* for negative instances, the plots in a sense confirm the theoretical (super)exponential worst-case time complexity. Indeed, the reader can see that the plots resemble the positive part of the graph of an exponential function. The growth of the function is fastest for *vata* and for positive instances of *trellis*. On the other hand, for *limi*, the growth is very slow, which makes the tool very attractive for the IFO verification.

We would like to point out that the results on random data, presented at the git repository, show the same results. There, we generated uniformly at random 300 IFO instances for each of $250, 500, 750, \ldots, 6000$ states, which resulted in 7200 random IFO instances.

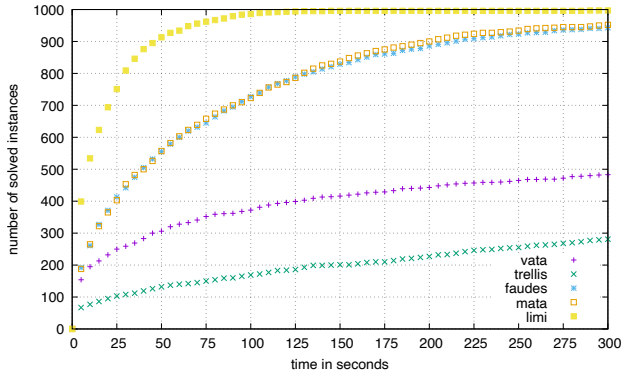
Quite an efficient behavior of the advanced inclusion-based tools, in particular of the *limi* tool, compared with the *trellis* tool, is of particular significance, namely if the reader realizes that the inputs for the inclusion-based tools are of size quadratically larger than the inputs for *trellis*. In many cases, they have millions of states and even more transitions.

[2] `https://fgdes.tf.fau.de/faudes/index.html`

[3] For technical problems and memory consumptions, we excluded *hkc* of Bonchi and Pous (2013) and *rabit* of Clemente and Mayr (2019).



(a) Negative instances.



(b) Positive instances.

Figure 2. Instances computed in five minutes. The x-axis displays the time (0–300 seconds), and the y-axis displays the number of solved instances in that time (0–994 for negative and 0–1001 for positive instances).

Finally, we would like to mention that the *trellis* and *faudes* tools implementing the textbook algorithms (Algorithm 1 and Algorithm 2) are on purpose without any optimizations. Our results thus in no way say anything about (in)efficiency of the `libFAUDES` library.

Acknowledgements

We gratefully acknowledge suggestions and comments of the anonymous referees.

References

Abdulla, P.A., Chen, Y., Holík, L., Mayr, R., Vojnar, T., 2010. When simulation meets antichains, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 158–174.

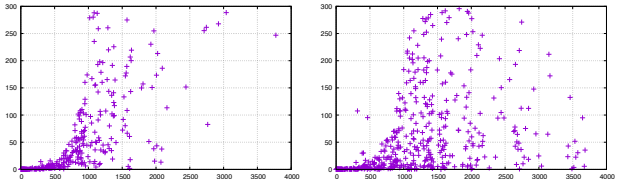
Balun, J., Masopust, T., Osička, P., 2023. Speed me up if you can: Conditional lower bounds on opacity verification, in: International Symposium on Mathematical Foundations of Computer Science, pp. 16:1–16:15.

Balun, J., Masopust, T., Osička, P., 2024. K-step opacity benchmarsk for discrete event systems. URL: https://apollo.inf.upol.cz:81/masopust/k-so-opacity-benchmarks. manuscript.

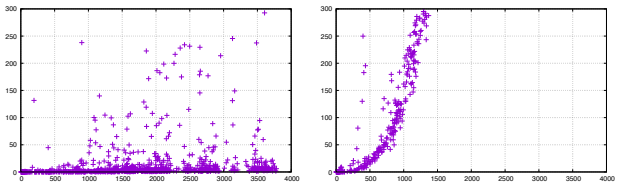
Bonchi, F., Pous, D., 2013. Checking NFA equivalence with bisimulations up to congruence, in: Symposium on Principles of Programming Languages, pp. 457–468.

Brzozowski, J.A., Fich, F.E., 1980. Languages of R-trivial monoids. Journal of Computer and Systems Sciences 20, 32–49.

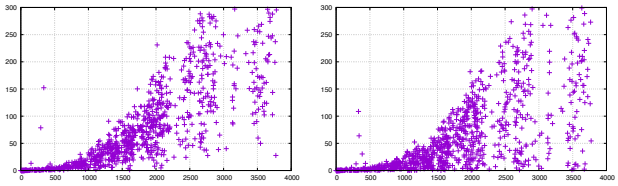
Černý, P., Clarke, E.M., Henzinger, T.A., Radhakrishna, A., Ryzhyk, L., Samanta, R., Tarrach, T., 2017. From non-preemptive to preemptive scheduling using synchronization synthesis. Formal Methods in System Design 50, 97–139.



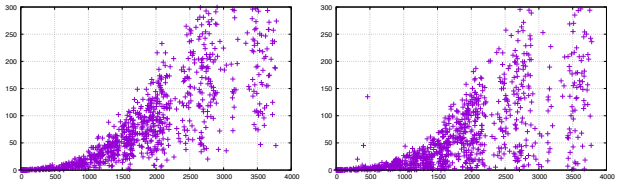
(a) vata



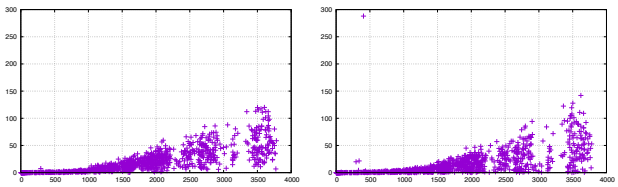
(b) trellis



(c) faudes



(d) mata



(e) limi

Figure 3. Time to solve negative (left) and positive (right) instances. The x-axis displays the size of the instance (0–4000 states), and the y-axis displays the time to solve the instance (0–300 s).

Chocholatý, D., Fiedor, T., Havlena, V., Holík, L., Hruska, M., Lengál, O., Síc, J., 2024. Mata: A fast and simple finite automata library, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 130–151.

Clemente, L., Mayr, R., 2019. Efficient reduction of nondeterministic automata with application to language inclusion testing. Logical Methods in Computer Science 15.

Dénes, J., 1966. On transformations, transformation-semigroups and graphs, in: Theory of Graphs. Proceedings of the Colloquium on Graph Theory, pp. 65–75.

Devadze, H., 1968. Generating sets of the semigroup of all binary relations on a finite set. Dokl. Akad. Nauk SSSR 12, 765–768. (Russian).

Feng, L., Wonham, W.M., 2010. On the computation of natural observers in discrete-event systems. Discrete Event Dynamic Systems 20, 63–102.

Hennessy, M., Milner, R., 1980. On observing nondeterminism and concurrency, in: International Colloquium on Automata, Languages and Programming, pp. 299–309.

Hivert, F., Mitchell, J.D., Smith, F.L., Wilson, W.A., 2021. Minimal generating sets for matrix monoids. arXiv:2012.10323.

Holzer, M., König, B., 2004. On deterministic finite automata and syntactic monoid size. Theoretical Computer Science 327, 319–347.

Hopcroft, J.E., Ullman, J.D., 1979. Introduction to Automata Theory, Languages and Computation. Addison-Wesley.

Howie, J., 1995. Fundamentals of Semigroup Theory. LMS monographs, Clarendon.

Impagliazzo, R., Paturi, R., 2001. On the complexity of k-sat. Journal of Computer and System Sciences 62, 367–375.

Jacob, R., Lesage, J., Faure, J., 2016. Overview of discrete event systems opacity: Models, validation, and quantification. Annual Reviews in Control 41, 135–146.

Jirásková, G., Masopust, T., 2012. On a structural property in the state complexity of projected regular languages. Theoretical Computer Science 449, 93–105.

Kim, K.H., Roush, F.W., 1978. Two-generator semigroups of binary relations. Journal of Mathematical Psychology 17, 236–246.

Konieczny, J., 2011. A proof of Devadze's theorem on generators of the semigroup of boolean matrices. Semigroup Forum 83, 281–288.

Krawetz, B., Lawrence, J., Shallit, J.O., 2005. State complexity and the monoid of transformations of a finite set. Intational Journal of Foundations of Computer Science 16, 547–563.

Krötzsch, M., Masopust, T., Thomazo, M., 2017. Complexity of universality and related problems for partially ordered NFAs. Information and Computation 255, 177–192.

Masopust, T., Krötzsch, M., 2021. Partially ordered automata and piecewise testability. Logical Methods in Computer Science 17.

Saboori, A., Hadjicostis, C.N., 2013. Verification of initial-state opacity in security applications of discrete event systems. Information Sciences 246, 115–132.

Salomaa, A., 1960. On the composition of functions of several variables ranging over a finite set. Annales Universitatis Turkuensis, Series A I 41.

Tange, O., 2023. Gnu parallel 20240122 (Frederik X). URL: https://doi.org/10.5281/zenodo.10558745.

Wintenberg, A., Blischke, M., Lafortune, S., Ozay, N., 2022. A general language-based framework for specifying and verifying notions of opacity. Discrete Event Dynamic Systems 32, 253–289.

Wong, K., 1998. On the complexity of projections of discrete-event systems, in: Workshop on Discrete Event Systems, Cagliari, Italy. pp. 201–206.

Wong, K., Wonham, W., 1996. Hierarchical control of discrete-event systems. Discrete Event Dynamic Systems 6, 241–273.

Wong, K.C., Wonham, W.M., 2004. On the computation of observers in discrete-event systems. Discrete Event Dynamic Systems 14, 55–107.

Wu, Y.C., Lafortune, S., 2013. Comparative analysis of related notions of opacity in centralized and coordinated architectures. Discrete Event Dynamic Systems 23, 307–339.