Received: 19 August 2022 | Revised: 5 July 2023 | Accepted: 10 July 2023

DOI: 10.1002/spe.3246

RESEARCH ARTICLE

WILEY

Quick Subset Construction

Michele Dusi | **Gianfranco Lamperti**

Department of Information Engineering, University of Brescia, Brescia, Italy

Correspondence
Gianfranco Lamperti, Department of Information Engineering, University of Brescia, Via Branze 38, Brescia, 25123, Italy.
Email: gianfranco.lamperti@unibs.it

Abstract

A finite automaton can be either deterministic (DFA) or nondeterministic (NFA). An automaton-based task is in general more efficient when performed with a DFA rather than an NFA. For any NFA there is an equivalent DFA that can be generated by the classical SUBSET CONSTRUCTION algorithm. When, however, a large NFA may be transformed into an equivalent DFA by a series of actions operating *directly* on the NFA, SUBSET CONSTRUCTION may be unnecessarily expensive in computation, as a (possibly large) deterministic portion of the NFA is regenerated as is, a waste of processing. This is why a conservative algorithm for NFA determinization is proposed, called QUICK SUBSET CONSTRUCTION, which progressively transforms an NFA into an equivalent DFA instead of generating the DFA from scratch, thereby avoiding unnecessary processing. QUICK SUBSET CONSTRUCTION is proven, both formally and empirically, to be equivalent to SUBSET CONSTRUCTION, inasmuch it generates exactly the same DFA. Experimental results indicate that, the smaller the number of repair actions performed on the NFA, as compared to the size of the equivalent DFA, the faster QUICK SUBSET CONSTRUCTION over SUBSET CONSTRUCTION.

KEYWORDS

determinization, finite automata, inward-oriented determinization, nondeterminism, subset construction

1 | INTRODUCTION

SUBSET CONSTRUCTION[1] is the classical algorithm that generates a *deterministic* finite automaton (DFA) that is equivalent to a given *nondeterministic* finite automaton (NFA). The need for NFA determinization is grounded on practical reasons: generally speaking, automaton-based tasks perform better with DFAs than they do with NFAs. A peculiarity of SUBSET CONSTRUCTION is generating an equivalent DFA from scratch, thereby leaving the input NFA untouched. That algorithm does not operate directly on the NFA, by fixing the points that are affected by nondeterminism; instead, a whole DFA is constructed starting from the initial state. This so-called *outward-oriented* approach to determinization makes sense as long as either the NFA is small or a large number of repair actions are required to remove the nondeterminism, as there is no point in fixing the NFA rather than generating a new DFA when the equivalent DFA differs from the NFA considerably. When, instead, a large NFA may be transformed into an equivalent DFA by a limited number of repair actions (as compared to the size of the DFA), building a DFA from scratch may be less than optimal because a

Abbreviations: DFA, Deterministic Finite Automaton; FA, Finite Automaton; NFA, Nondeterministic Finite Automaton.

(possibly large) portion of the NFA that is deterministic already is unnecessarily processed. Put another way, a large NFA may possibly be determinized more efficiently than SUBSET CONSTRUCTION does by fixing directly the NFA instead of generating an equivalent DFA from scratch. Fixing an NFA means applying a series of repair actions on the nondeterministic transition function that progressively transform the NFA into exactly the same equivalent DFA produced by SUBSET CONSTRUCTION. This alternative technique to determinization of NFAs is said to be *inward-oriented*.

Consider, for example, an autonomous agent (robot) that is designed to learn by experience the environment in which it is being operated. To this end, the robot may construct incrementally a model of the environment based on a DFA, where states represent configurations of the environment, whereas transitions indicate actions of the robot that cause a change in the environment configuration. Given the current instance of the DFA, an interaction of the robot with the environment is bound to extend the DFA with new transitions and states. If such an extension comes with nondeterminism, the DFA stops being deterministic, thereby becoming an NFA. This NFA may be very large (certainly so if the previous DFA is large), with nondeterminism being confined to the newly-created transitions. Now, since the model of the environment in which the robot is being operated is supposed to be a DFA, just-in-time determinization of the NFA is required in order to update the deterministic model of the environment. Determinization by SUBSET CONSTRUCTION would generate the new equivalent DFA from scratch, however, even if the new DFA is likely to be almost equal to the previous DFA. From a computational point of view, regenerating the DFA from scratch on the fly may be impractical, especially so if performed under stringent time constraints.

A more practical approach may be to update directly the NFA by fixing the points in which the transition function becomes nondeterministic, thereby avoiding the unnecessary regeneration of the deterministic portion of the NFA. This alternative approach to NFA determinization is substantiated by a novel algorithm, named QUICK SUBSET CONSTRUCTION, which is the subject of this article. Unlike SUBSET CONSTRUCTION, the proposed algorithm progressively transforms the NFA into an equivalent DFA that, remarkably, is identical to the DFA generated from scratch by SUBSET CONSTRUCTION. In this sense, QUICK SUBSET CONSTRUCTION is equivalent to SUBSET CONSTRUCTION, even though the same DFA is obtained in a very different mode.

QUICK SUBSET CONSTRUCTION was not conceived overnight: it is the result of a series of algorithms designed for a specific research domain of artificial intelligence named *model-based diagnosis*.[2] Specifically, the problem of incremental determinization of finite automata was addressed in monitoring-based diagnosis of active systems,[3] which requires the incremental determinization of an NFA that changes over time.[4-7]

To our knowledge, in the literature there is no inward-oriented determinization algorithm. Indeed, starting from SUBSET CONSTRUCTION, all the alternative determinization algorithms are invariably outward-oriented, where the equivalent DFA is generated from scratch.* Perhaps, in this respect, the closest algorithm to QUICK SUBSET CONSTRUCTION is SUBSET RESTRUCTURING,[7] which was designed for the determinization of *mutating* automata. A mutating finite automaton is an NFA that changes its morphology over discrete time by a sequence of *mutations*, where a mutation causes some changes in the transition function of an NFA, such as the insertion and/or removal of some transitions as well as involved states. This results in a temporal sequence of NFAs, namely $\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_i, \ldots$, where $\mathcal{N}_0$ is the initial NFA, whereas each subsequent NFA results from a mutation of the preceding NFA. The idea is to generate *on the fly* an equivalent DFA of each NFA, say $\mathcal{D}_{i+1}$ that is equivalent to $\mathcal{N}_{i+1}$, based on the DFA $\mathcal{D}_i$ (generated previously), that is equivalent to $\mathcal{N}_i$, and the difference (in the transition function) between $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$. Instead of generating $\mathcal{D}_{i+1}$ from scratch as SUBSET CONSTRUCTION does, $\mathcal{D}_{i+1}$ is obtained by updating $\mathcal{D}_i$ while accounting for the mutation leading $\mathcal{N}_i$ to $\mathcal{N}_{i+1}$. On the other hand, SUBSET RESTRUCTURING cannot cope with the determinization of any given NFA outside the narrow context of mutations, which is instead possible for QUICK SUBSET CONSTRUCTION, which can therefore be regarded as a general-purpose algorithm for NFA determinization, exactly like SUBSET CONSTRUCTION, although based on a very different approach.

Specifically, the main contributions of the current paper are:

1. A detailed presentation of the QUICK SUBSET CONSTRUCTION determinization algorithm;
2. A proof of equivalence of QUICK SUBSET CONSTRUCTION, which demonstrates that the resulting DFA is identical to the DFA generated from scratch by SUBSET CONSTRUCTION;
3. An implementation of a software framework (available online as open source) for experimenting with QUICK SUBSET CONSTRUCTION, as well as for generating pseudo-random test cases based on given configuration parameters;
4. A variety of experimental results that compare QUICK SUBSET CONSTRUCTION with SUBSET CONSTRUCTION.

*Even the algorithm proposed by Brzozowski's,[8] which generates the minimal DFA that is equivalent to a given NFA/DFA, exploits SUBSET CONSTRUCTION as is, thereby operating outward the input NFA/DFA.

In the rest of the article, Section 2 recalls the notion of a finite automaton, specifically DFA and NFA. Section 3 presents the classical outward-oriented technique for NFA determinization by SUBSET CONSTRUCTION. Section 4 illustrates in detail the inward-oriented technique for NFA determinization by QUICK SUBSET CONSTRUCTION. The equivalence of SUBSET CONSTRUCTION and QUICK SUBSET CONSTRUCTION is formally proven in Section 5. Experimental results are presented in Section 6, along with a discussion on the convenience of using QUICK SUBSET CONSTRUCTION rather than SUBSET CONSTRUCTION. Conclusions are drawn in Section 7.

2 | FINITE AUTOMATA

A *finite automaton* (FA) is a mathematical formalism for the modeling of a variety of systems,[9] which can be either abstract or concrete. Abstract formal systems modeled as FAs are designed to solve problems like pattern matching based on regular expressions,[10] language recognizers and translators,[11] as well as analysis of protein sequences.[12] For instance, a lexical analyzer, which is designed to recognize the words of a language, can be implemented as an FA, where the input symbols are the characters of a given alphabet, while a state represents a prefix of the word being recognized, for example, an identifier in a programming language. The (possibly infinite) set of strings that can be recognized by an FA is called a *regular language*. Every regular language can be specified by a regular expression, and each language specified by a regular expression is regular and can be recognized by an FA. Put another way, FAs and regular expressions share the same expressive power.

As pointed out, an FA can also model a concrete system, which is typically *discrete* and *dynamic*. A discrete system is characterized by input variables and system states that can be represented by discrete values. A dynamic (or time-varying) system evolves over time from one state to another. Moreover, in a system modeled by an FA, both the domain of input symbols and the set of states are finite. In both control theory and artificial intelligence, FAs are exploited to model a class of physical systems called *discrete-event systems*.[13] Once modeled, a discrete-event system can be subjected to specific engineering tasks, including monitoring and diagnosis.[3,14-26]

Formally, an FA can be either *deterministic* (DFA) or *nondeterministic* (NFA). Specifically, a DFA is a 5-tuple,

$$\mathcal{A} = (\Sigma, Q, \delta, q_0, F), \tag{1}$$

where Σ is the alphabet (a finite set of symbols called hereafter *labels*), Q is the (finite) set of states, δ is the transition function, $\delta : Q_\Sigma \mapsto Q$, where $Q_\Sigma \subseteq Q \times \Sigma$, q_0 is the initial state, and F is the set of final states, where $F \subseteq Q$. Determinism in $\mathcal{A}$ comes from the peculiarity of the transition function mapping each pair $(q, \ell) \in Q_\Sigma$ into a *single* state q', namely $\delta(q, \ell) = q'$.

Example 1 (DFA). Centered in Figure 1 is a graphical representation of a DFA, which includes six states and eight transitions, where A is the initial state and G is the (unique) final state. Determinism is grounded on the fact that all the transitions exiting the same state have different symbols of the alphabet, in other words, the transition function maps each pair in Q_Σ into a single state, for instance, $\delta(D, a) = E$ and $\delta(D, b) = G$.

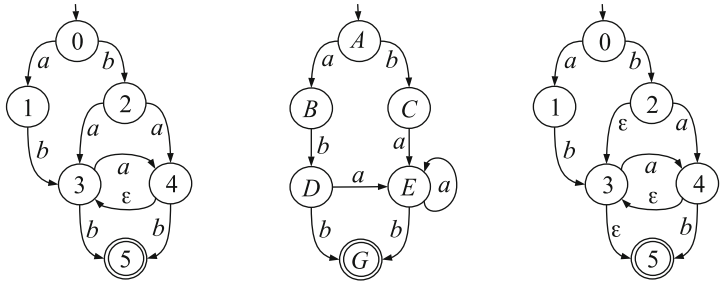

FIGURE 1 An NFA (left) and an equivalent DFA (center), which recognize the regular language $(aba^*|ba^+)\ b$. A slight variation of the left NFA is displayed on the right side.

An NFA is defined almost identically to a DFA, with the exception of the transition function. Formally, an NFA is a 5-tuple,

$$\mathcal{N} = (\Sigma, Q, \delta, q_0, F), \tag{2}$$

where Σ, Q, q_0, and F are defined as in Equation (1), whereas the transition function δ is defined differently, $\delta : Q^{\varepsilon}_{\Sigma} \mapsto 2^{Q}$, where $Q^{\varepsilon}_{\Sigma} \subseteq Q \times (\Sigma \cup \{\varepsilon\})$, with ε being the *empty* symbol ($\varepsilon \notin \Sigma$). Nondeterminism in $\mathcal{N}$ lies in the transition function, which, on the one hand, involves the empty symbol ε, which allows for *spontaneous* state transitions and, on the other, it maps a pair (q, ℓ), where $q \in Q$ and $\ell \in \Sigma \cup \{\varepsilon\}$, into a *set* of states. Intuitively, an NFA may be in several states when recognizing a string of symbols, which is impossible for a DFA, which is always in a single state during processing.[†]

Roughly, the regular language recognized by an FA, either deterministic or nondeterministic, is the set of strings (composed of the symbols in the alphabet) that we obtain by traversing all paths of transitions from the initial state to a final state, where the instances of the empty symbol ε are immaterial. Two FAs are *equivalent* if they share (recognize) the same regular language. From a practical stand point, however, equivalence is not synonymous with equality in computation. Given an NFA and an equivalent DFA, the processing based on the NFA is in general more complex than the processing based on the DFA. In lexical analysis, for instance, the recognition of a string based on a DFA requires the processing just to keep track of one state, namely the state computed by the transition function $\delta(q, \ell)$, where q is the current state and ℓ is the next character in input. By contrast, the same recognition based on an NFA requires the lexical analyzer to keep track of a *set* of states. In fact, since the transition function maps one state to several states, the NFA may be in several states during processing, namely in a set $\overline{Q} = \{q_1, \ldots, q_k\}$. Hence, the recognition of the next character ℓ in input will move the NFA to a set of states $\overline{Q}_{\ell} \cup \overline{Q}^{*}_{\ell}$, where

$$\overline{Q}_{\ell} = \bigcup_{i \in [1\,..\,k]} \delta(q_i, \ell), \tag{3}$$

and $\overline{Q}^{*}_{\ell}$ is the set of states that are reachable from any state in $\overline{Q}_{\ell}$ by paths of ε-transitions. We have, therefore, good reasons to prefer an equivalent DFA over an NFA, especially considering that for any NFA there is always an equivalent DFA, more generally, several (possibly an infinite number of) equivalent DFAs.

Example 2 (Equivalent FAs). With reference to Figure 1, note how the DFA displayed in the center is equivalent to the NFA on the left side. In fact, they both recognize the same regular language, which is defined by the following regular expression:

$$(aba^{*}|ba^{+})\ b. \tag{4}$$

For instance, the string *bab* is recognized by the DFA via the following sequence of transitions: $\delta(A, b) = C$, $\delta(C, a) = E$, $\delta(E, b) = G$. Since G (the last state) is final, the string is recognized. The same string is recognized by the equivalent NFA on the left of Figure 1 as follows: matching the first character moves the NFA to the singleton $\delta(0, b) = \{2\}$; next, matching the second character moves the NFA to the set of states $\delta(2, a) = \{3, 4\}$; finally, matching the third character moves the NFA to the set of states $\delta(3, b) \cup \delta(4, b) = \{5\}$. Since this last set contains a final state, the string is recognized.

Before presenting the classical algorithm for NFA determinization, we introduce a little handy terminology. Given a state s of an FA and a label ℓ of the alphabet, a transition mapping (s, ℓ) is called an *ℓ-transition*. Let n be a state of an NFA $\mathcal{N}$. The *ε-closure* of n, denoted ε-closure$(n, \mathcal{N})$, is the set of states in $\mathcal{N}$ that is composed of n and all the states that are reachable from n by paths of ε-transitions. The *ε-closure* of a set $\overline{N}$ of states in $\mathcal{N}$ is defined as

$$\varepsilon\text{-closure}(\overline{N}, \mathcal{N}) = \bigcup_{n \in \overline{N}} \varepsilon\text{-closure}(n, \mathcal{N}). \tag{5}$$

[†]In the literature, there is a distinction between an NFA and an ε-NFA, where the former does not include ε-transitions, while the latter does. In this article, we do not make that distinction and consider a more general notion of an NFA, which may or may not include ε-transitions.

Example 3 (ε-closure). With reference to the NFA $\mathcal{N}$ displayed on the right of Figure 1 (a slight variation of the NFA shown on the left side of the figure), we have ε-closure$(2, \mathcal{N}) = \{2, 3, 5\}$ and ε-closure$(\{3, 4\}, \mathcal{N}) = \{3, 4, 5\}$.

Let n be a state of an NFA $\mathcal{N}$ and let ℓ be a label in the alphabet of $\mathcal{N}$. The *ℓ-mapping* of n, denoted ℓ-mapping$(n, \mathcal{N})$, is the ε-closure of the set of states generated by the transition function when applied to (n, ℓ). The *ℓ-mapping* of a set $\overline{N}$ of states in $\mathcal{N}$, is defined as

$$\ell\text{-mapping}(\overline{N}, \mathcal{N}) = \bigcup_{n \in \overline{N}} \ell\text{-mapping}(n, \mathcal{N}). \tag{6}$$

Example 4 (ℓ-mapping). Considering the NFA $\mathcal{N}$ displayed on the right of Figure 1, we have a-mapping$(2, \mathcal{N}) = \{3, 4, 5\}$ and b-mapping$(\{0, 1\}, \mathcal{N}) = \{2, 3, 5\}$.

3 | OUTWARD-ORIENTED DETERMINIZATION BY SUBSET CONSTRUCTION

The classical algorithm for NFA determinization is SUBSET CONSTRUCTION.[1] The reason for this name stems from the mode in which each state of the equivalent DFA is constructed, namely as a *subset* of the NFA states. Moreover, if the NFA includes m states, in the worst case, the DFA will include $2^m - 1$ states, which is the number of possible subsets of NFA states (empty set excluded). In other words, in the worst case, the complexity of the DFA generated by SUBSET CONSTRUCTION is exponential in the number of NFA states. In practice, however, this is very pessimistic, as possibly a large number of subsets of NFA states are not involved in the DFA constructed.

The *modus operandi* of SUBSET CONSTRUCTION is outward-oriented: it maps an NFA to an equivalent DFA that is constructed from scratch, which, for this reason, is said to be *SC-equivalent* to the input NFA. Although there are in general several DFAs that are equivalent to a given NFA, nonetheless there exists just one *minimal* equivalent DFA, namely a DFA including the minimum number of states. Several algorithms exist for generating the minimal DFA that is equivalent to a given DFA.[8,27-34] Therefore, even if the *SC*-equivalent DFA is not minimal, it can be minimized automatically.

The pseudocode of SUBSET CONSTRUCTION is listed in Algorithm 1 (lines 1–23). It takes as input an NFA $\mathcal{N}$ and generates as output a DFA $\mathcal{D}$ that is equivalent to $\mathcal{N}$. For practical reasons, the algorithm makes a distinction between the identifier d of a state of the output DFA and the subset of states marking d, called the *extension* of d, denoted $\|d\|$. It exploits a stack onto which the newly-created states are put. The idea is to pop one state at a time from the stack and to generate the relevant transition function, which may cause the creation of new states. The processing terminates when the stack becomes empty, that is, when the transition function of all states has been completed. Specifically, the set D of states, the set δ_{d} of transitions, where a transition $\delta_{\mathrm{d}}(d, \ell) = d'$ is denoted as an arc $\langle d, \ell, d' \rangle$, and the set F_{d} of final states are initialized as empty (line 5). In line 6, the initial state d_0 is inserted into D, where $\|d_0\| = \varepsilon$-closure$(n_0, \mathcal{N})$. In other words, the extension of d_0 includes the initial state n_0 of $\mathcal{N}$ plus the states of $\mathcal{N}$ that are reachable from n_0 by ε-transitions only. If the extension of d_0 includes a final state of $\mathcal{N}$, then d_0 is inserted into the set of final states of $\mathcal{D}$ also (line 7). Then, d_0 is pushed onto the stack (line 8). Afterwards, the loop in lines 9–22 is executed until the stack of states becomes empty. At each iteration, a state d is popped from the stack (line 10) and the transitions exiting d in $\mathcal{D}$ are generated within the nested loop in lines 11–21. To this end, each label ℓ of the alphabet Σ relevant to a transition exiting an NFA state in $\|d\|$ is considered (line 11). In line 12, a set $\overline{N}$ of states in $\mathcal{N}$ is computed as ℓ-mapping$(\|d\|, \mathcal{N})$, that is, the set of states of $\mathcal{N}$ that are reached by the ℓ-transitions exiting the states in $\|d\|$ plus the states in $\mathcal{N}$ that are reachable from these states by ε-transitions only. $\overline{N}$ is the extension of the target state of the ℓ-transition exiting d in $\mathcal{D}$. Hence, if there is no such a state (line 13), then it is created, namely d' with extension $\overline{N}$ (line 14) and possibly added to the set of final states provided that $\|d'\|$ includes at least one final state of $\mathcal{N}$ (line 15); then, since it is a newly-created state, d' is pushed onto the stack (line 16). If, instead, that state exists already, then it is referenced as d' (line 18). In either case, a new transition $\langle d, \ell, d' \rangle$ is eventually created in $\mathcal{D}$ (line 20). Notice that, being the set N of states in $\mathcal{N}$ finite, the set of states that can be generated for $\mathcal{D}$ is finite also, as it is bounded by the powerset of N; hence, sooner or later, SUBSET CONSTRUCTION terminates and $\mathcal{D}$ is in fact a DFA, namely the *SC*-equivalent DFA of $\mathcal{N}$.

Example 5 (SUBSET CONSTRUCTION). Detailed in Figure 2 is the generation, performed by SUBSET CONSTRUCTION, of the DFA that is SC-equivalent to the NFA depicted on the left side of Figure 1. Specifically,

Algorithm 1. *Subset Construction*

1: **function** SUBSET CONSTRUCTION($\mathcal{N}$) $\rightarrow \mathcal{D}$
2: **input:** $\mathcal{N} = (\Sigma, N, \delta_n, n_0, F_n)$: an NFA
3: **output:** $D = (\Sigma, D, \delta_d, d_0, F_d)$: a DFA that is equivalent to $\mathcal{N}$, namely the *SC*-equivalent DFA of $\mathcal{N}$
4: **begin**
5: Initialize D, δ_d, and F_d as empty sets
6: Insert the initial state d_0 into D, where $\|d_0\| = \varepsilon$-closure$(n_0, \mathcal{N})$
7: **if** $\|d_0\|$ includes a final state of $\mathcal{N}$ **then** insert d_0 into F_d **end if**
8: Initialize a stack of states by including d_0 only
9: **repeat**
10: Pop a state d from the stack
11: **for all** $\ell \in \Sigma$ such that there is a state $n \in \|d\|$ for which $\delta_n(n, \ell)$ is defined **do**
12: $\bar{N} \leftarrow \ell$-mapping$(\|d\|, \mathcal{N})$
13: **if** there is no state $d' \in D$ such that $\|d'\| = \bar{N}$ **then**
14: Insert a new state d' into D, where $\|d'\| = \bar{N}$
15: **if** $\|d'\| \cap F_n \neq \emptyset$ **then** insert d' into F_d **end if**
16: Push d' onto the stack
17: **else**
18: Let $d' \in D$ such that $\|d'\| = \bar{N}$
19: **end if**
20: Insert a new arc $\langle d, \ell, d' \rangle$ into δ_d
21: **end for**
22: **until** the stack is empty
23: **end function**

each intermediate instance of the DFA is associated with the content of the stack (placed on top of each instance), which grows from left to right; hence, the rightmost state (in bold) represents the top of the stack, that is, the next state to be processed. At the beginning of the main loop (lines 9–22), the DFA is composed of the initial state only, namely A, which is also the unique element in the stack. Then, state A is popped from the stack and the relevant transition function is materialized by means of two exiting transitions, while the two newly-created states, namely B and C, are inserted onto the stack. The algorithm continues popping one state from the stack and generating the relevant transition function, while putting possible new states onto the stack. For instance, the processing of state C gives rise to the creation of the new state E, where $\|E\| = a$-mapping$(\{2\}) = \{3, 4\}$. When the stack becomes empty (right side of Figure 2), the DFA is complete. Note that the resulting DFA is identical to the DFA displayed on the center of Figure 1, except that each state is marked with the relevant extension, a by-product of the mode in which SUBSET CONSTRUCTION identifies the DFA states. Once the DFA is generated, however, all the extensions may be removed and the states are simply identified by chosen symbols (in our example, capital letters).

Since SUBSET CONSTRUCTION is a function, for each NFA $\mathcal{N}$ there is one and only one DFA that is *SC*-equivalent to $\mathcal{N}$, albeit, generally speaking, several DFA might exist that are equivalent to $\mathcal{N}$.

4 | INWARD-ORIENTED DETERMINIZATION BY QUICK SUBSET CONSTRUCTION

In order to generate an equivalent DFA, the SUBSET CONSTRUCTION algorithm does not operate directly on the NFA: the DFA is constructed from scratch leaving the input NFA untouched. Yet, this outward-oriented approach is not strictly necessary to the determinization task. In fact, the same equivalent DFA can also be obtained using a series of repair actions operating directly on the input NFA: the nondeterminism is progressively removed from the NFA by fixing the transition function wherever necessary, thereby eventually obtaining the *SC*-equivalent DFA.

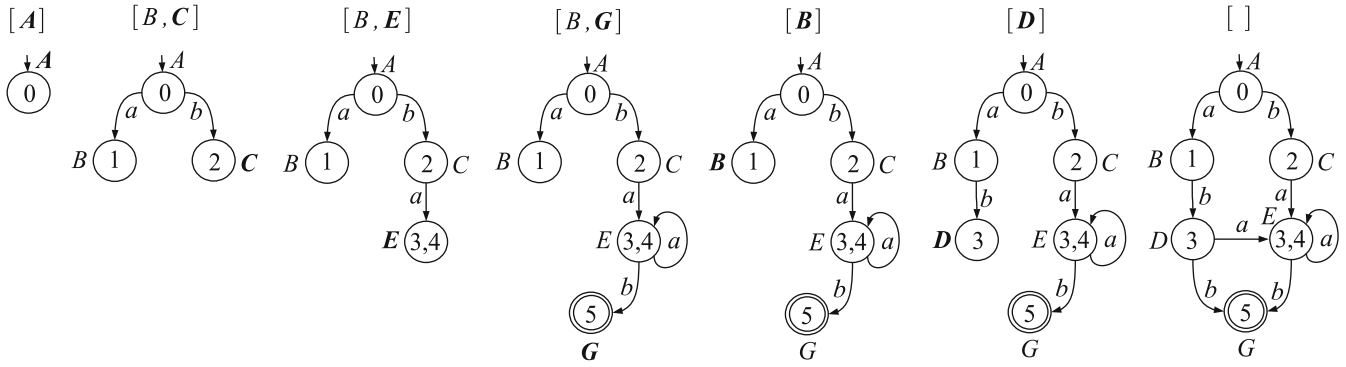

FIGURE 2 Generation by SUBSET CONSTRUCTION of the DFA *SC*-equivalent to the NFA displayed on the left of Figure 1. Displayed on top of each intermediate graph is the content of the stack (growing from left to right).

This idea is substantiated by an alternative algorithm for NFA determinization, called QUICK SUBSET CONSTRUCTION[‡] To this end, each point of nondeterminism in the transition function of the NFA is considered and a specific repair action is applied in order to remove the nondeterminism. Nondeterminism occurs when a state is exited either by an ε-transition or by several transitions with the same label.

4.1 | Singularities

In order to capture all the nondeterminism points, the relevant states of the NFA are marked with a set of labels in $\Sigma \cup \{\varepsilon\}$, where Σ is the alphabet of the NFA. The initial marking is based on three rules:

1. If an ε-transition exits the initial state n_0 of the NFA, then n_0 is marked with ε.
2. If there is a transition $\langle n, \ell, n' \rangle$, where $\ell \neq \varepsilon$, and n' is exited by an ε-transition, then n is marked with ℓ.
3. If a state n is exited by several ℓ-transitions, where $\ell \neq \varepsilon$, then n is marked with ℓ.

If a label ℓ marks a state n, then (n, ℓ) is called a *singularity*. Since the collection of labels marking a state is an ordered set, no duplication of the same singularity is allowed. A singularity initially marking the NFA indicates that the transition function mapping (n, ℓ) results in several states, in other words, nondeterminism arises; consequently, a repair action is required in order to fix that singularity. Fixing a singularity may cause the creation of additional singularities, which indicate that the transition function of the relevant states needs to be either adjusted or completed with new transitions.

Any given NFA is therefore associated with a set of *initial singularities*. The notion of an initial singularity is exploited for providing a precise measure of the quantity of nondeterminism affecting an NFA (Definition 1).[§]

> **Definition 1** (quantity of nondeterminism). The quantity of nondeterminism in an NFA is the number of initial singularities.

Definition 1 should not be misleading, however: as shown in Section 6.6, the quantity of nondeterminism in an NFA is generally unrelated to the number of singularities actually processed, which, in some circumstances, can be far larger than the number of initial singularities. That is, a small quantity of nondeterminism may translate into a large number of repair actions.

The order in which singularities are processed is important. Based on our experience, choosing the singularities randomly may cause the disconnection of the resulting DFA and/or the nontermination of the algorithm. In order to avoid both disconnection and nontermination, singularities are sorted based on the *level* of the states. The level of a state s in an FA with initial state s_0, denoted $\lambda(s)$, is the minimum number of transitions (including ε-transitions) that connect s_0 with s. Operationally, the level of a state can be defined inductively by the following two rules:

[‡]In that respect, QUICK SUBSET CONSTRUCTION is equivalent to SUBSET CONSTRUCTION (cf. Section 5).

[§]Definition 1 is only instrumental within the narrow scope of this article: the same definition may be questionable if extrapolated from that context.

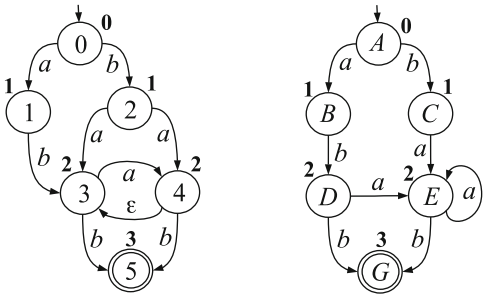

FIGURE 3 NFA (left) and DFA (right), with states marked with corresponding levels (cf. Figure 1).

1. $\lambda(s_0) = 0$.
2. For any state $s \neq s_0$ having parent states[¶] $s_1, \ldots, s_k$, $\lambda(s) = \min\{\lambda(s_1), \ldots, \lambda(s_k)\} + 1$.

Example 6 (level). Shown in Figure 3 are the NFA (left) and DFA (right) displayed in Figure 1, where each state is marked with the relevant level.

Since the level imposes only a partial order on states, as several states may have the same level, in order for QUICK SUBSET CONSTRUCTION to behave deterministically,[#] the singularities (q, ℓ) are totally ordered based not only on $\lambda(q)$ but also on the natural (ascending) order of the identifiers q and ℓ (with the label ε being in first position). Therefore, the singularities (q, ℓ) are sorted based first on $\lambda(q)$, then on the identifier of q, and finally on ℓ. The resulting (sorted) list of singularities is called a *singularity list*. Since processing a singularity may cause the insertion/removal of transitions, with possible changes in the level of states, the position of each singularity within the singularity list may change during processing.

As every state q in the FA being manipulated by QUICK SUBSET CONSTRUCTION is marked with a nonempty subset of the states of the NFA, as for SUBSET CONSTRUCTION, the new algorithm makes a distinction between an identifier q and the subset of states marking q, namely the extension $\|q\|$. Note that the extension of a state q can be changed by the repair actions, whereas the identifier q cannot. For formal reasons, we widen the notion of extension to an NFA state n, namely $\|n\| = \{n\}$. Also, given a set $\overline{Q}$ of states in the automaton being manipulated by QUICK SUBSET CONSTRUCTION, $\|\overline{Q}\|$ denotes the union of the extensions of the states in $\overline{Q}$, that is, a subset of the states of the input NFA.

4.2 | QUICK SUBSET CONSTRUCTION

The pseudocode of QUICK SUBSET CONSTRUCTION is listed in Algorithm 2 (lines 1–47).[‖] The algorithm exploits five auxiliary functions/procedures, namely UNSAFE, ENLARGE, NEW, LEVEL, and UNIFY (detailed in Section 4.3). QUICK SUBSET CONSTRUCTION takes as input an NFA $\mathcal{N}$ and generates a DFA that is *SC*-equivalent to $\mathcal{N}$. To this end, first a copy $\mathcal{Q}$ of $\mathcal{N}$ is created along with its initial singularities (line 5). Upon the termination of the algorithm, $\mathcal{Q}$ has been transformed into the equivalent DFA. For the determinization of $\mathcal{Q}$, the singularities are considered one by one within the singularity list. The repair actions associated with the singularity considered, however, depend on a relevant *scenario*. Three scenarios are defined, called $\mathcal{S}_0$, $\mathcal{S}_1$, and $\mathcal{S}_2$, respectively.

Scenario $\mathcal{S}_0$ (lines 6–19) occurs for the singularity (q_0, ε) only, called the ε-singularity, where q_0 is the initial state of $\mathcal{Q}$. This means that q_0 is exited by at least one ε-transition. Since, the ε-singularity is in the first position in the singularity list, this is the first singularity that is processed. Besides, since no singularity (q, ε) can be generated subsequently, neither for the initial state nor for any other state, the processing of the ε-singularity is placed upfront in the algorithm, before the main loop (lines 20–46), in which scenarios $\mathcal{S}_1$ and $\mathcal{S}_2$ are possibly considered several times. Roughly, the repair actions associated with (q_0, ε) remove the ε-transitions exiting q_0 while enlarging the extension of q_0. First the sets $\overline{Q}$, $\bar{U}$, and $\overline{N}$

[¶]A state $\bar{s}$ is a parent of a state s when there is a transition from $\bar{s}$ to s, namely $\langle \bar{s}, \ell, s \rangle$.

[#]The deterministic behavior of QUICK SUBSET CONSTRUCTION should not be confused with the deterministic function of a DFA: it means that QUICK SUBSET CONSTRUCTION always performs the same sequence of actions for the same input NFA to generate the *SC*-equivalent DFA.

[‖]The processing of final states is omitted: as in SUBSET CONSTRUCTION, a DFA state is final when its extension includes an NFA that is final.

Algorithm 2. *Quick Subset Construction*

function QUICK SUBSET CONSTRUCTION($\mathcal{N}$) $\to \mathcal{Q}$
  input: $\mathcal{N} = (\Sigma, N, \delta_n, n_0, F_n)$: an NFA
  output: $\mathcal{Q} = (\Sigma, Q, \delta_q, q_0, F_q)$: a DFA that is equivalent to $\mathcal{N}$, namely the *SC*-equivalent DFA of $\mathcal{N}$
begin
  Generate a copy $\mathcal{Q}$ of $\mathcal{N}$ and mark it with the initial singularities
  if there is a singularity (q_0, ε) **then** ⟨ *Scenario* $\mathcal{S}_0$*: lines 6–19* ⟩
    $\bar{Q} \leftarrow \varepsilon$-closure$(q_0, \mathcal{Q})$, $\bar{U} \leftarrow \{\bar{q} \mid \bar{q} \in \bar{Q}, \text{UNSAFE}((q_0, \varepsilon), \bar{q})\}$, $\bar{N} \leftarrow \|\bar{Q}\|$
    ENLARGE$(q_0, \bar{N})$
    Remove from δ_q all the ε-transitions exiting q_0
    for all transition $\langle u, \ell, q\rangle \in \delta_q$ where $u \in \bar{U}, \ell \neq \varepsilon,$ and $q \notin \bar{U}$ **do**
      Insert a transition $\langle q_0, \ell, q\rangle$ into δ_q, LEVEL$([(q, 1)])$
    end for
    for all transition $\langle q, \ell, u\rangle \in \delta_q$ where $q \notin \bar{U}$ and $u \in \bar{U}$ **do**
      Remove $\langle q, \ell, u\rangle$ from δ_q
      if $\ell \neq \varepsilon$ **then**generate the singularity (q, ℓ) **end if**
    end for
    Remove from $\mathcal{Q}$ all the states in $\bar{U}$ and the relevant entering/exiting transitions
    Remove (q_0, ε) from the singularity list
  end if
  while the singularity list is not empty, with (q, ℓ) being the singularity in first position **do**
    $\bar{N} \leftarrow \ell$-mapping$(\|q\|, \mathcal{N})$
    if no ℓ-transition exits q **then** ⟨ *Scenario* $\mathcal{S}_1$*: lines 22–28* ⟩
      if there is a state $q' \in Q$ where $\|q'\| = \bar{N}$ **then**
        Insert a transition $\langle q, \ell, q'\rangle$ into δ_q, LEVEL$([(q', \lambda(q) + 1)])$
      else
        $q' \leftarrow$ NEW$(\bar{N})$
        Insert a transition $\langle q, \ell, q'\rangle$ into δ_q and set the level $\lambda(q') = \lambda(q) + 1$
      end if
    else if several ℓ-transitions exit q **or** ⟨ *Scenario* $\mathcal{S}_2$*: lines 29–43* ⟩
        (either the only transition $\langle q, \ell, q'\rangle$ is such that $\|q'\| \neq \bar{N}$ **or** q' is exited by an ε-transition) **then**
      $\bar{Q} \leftarrow \ell$-*mapping*$(q, \mathcal{Q})$, $\bar{U} \leftarrow \{\bar{q} \mid \bar{q} \in \bar{Q}, \text{UNSAFE}((q, \ell), \bar{q})\}$
      $q' \leftarrow$ NEW$(\bar{N})$
      Remove from δ_q all the ℓ-transitions exiting q
      for all transition $\langle u, \ell', q''\rangle \in \delta_q$ where $u \in \bar{U}, \ell' \neq \varepsilon,$ and $q'' \notin \bar{U}$ **do**
        Insert a transition $\langle q', \ell', q''\rangle$ into δ_q
      end for
      for all transition $\langle q'', \ell', u\rangle \in \delta_q$ where $u \in \bar{U}$ and $q'' \notin \bar{U}$ **do**
        Remove $\langle q'', \ell', u\rangle$ from δ_q
        if $\ell' \neq \varepsilon$ **then** generate a singularity (q'', ℓ') **end if**
      end for
      Remove from $\mathcal{Q}$ all the states in $\bar{U}$ and the relevant entering/exiting transitions
      Insert a transition $\langle q, \ell, q'\rangle$ into δ_q, LEVEL$([(q', \lambda(q) + 1)])$
      if there is a state $q'' \in Q, q'' \neq q'$, where $\|q''\| = \|q'\|$ **then** UNIFY(q', q'') **end if**
    end if
    Remove (q, ℓ) from the singularity list
  end while
end function

are determined (line 7), where $\overline{Q}$ is the ε-closure of q_0, $\bar{U}$ is the set of the *unsafe* states in $\overline{Q}$, and $\overline{N}$ is the set of NFA states involved in $\overline{Q}$. Intuitively, a state q is unsafe if the removal of the ε-transitions may cause q to be no longer reachable from the initial state, in other words, $\mathcal{Q}$ may become disconnected. To know whether a state is unsafe, the auxiliary function UNSAFE is called (Algorithm 3, Section 4.3.1). Next, the extension of q_0 is expanded to $\overline{N}$ (line 8) by means of the ENLARGE auxiliary procedure (Algorithm 4, Section 4.3.2), which may also create new singularities for q_0. Then, the ε-transitions exiting q_0 are removed (line 9). Next, for each transition $\langle u, \ell, q\rangle$ in $\mathcal{Q}$ where u is unsafe, $\ell \neq \varepsilon$, and q is not in the set of unsafe states (possibly outside the set $\overline{Q}$), a transition $\langle q_0, \ell, q\rangle$ is created (lines 10–12), while the level of q and successive states is possibly updated by the LEVEL auxiliary procedure (Algorithm 6, Section 4.3.4). These new transitions prevent $\mathcal{Q}$ from becoming disconnected owing to the subsequent removal of transitions. Subsequently, every transition $\langle q, \ell, u\rangle$ exiting a state that is not unsafe (possibly outside the set $\overline{Q}$) and entering an unsafe state is removed and, if $\ell \neq \varepsilon$, a singularity (q, ℓ) is created (lines 13–16). This is required in order to avoid creating dangling transitions resulting from the subsequent removal of the unsafe states. Eventually, the unsafe states and relevant transitions are removed from $\mathcal{Q}$ (line 17), as well as the ε-singularity from the singularity list (line 18).

Example 7 (Scenario $\mathcal{S}_0$). Shown in Figure 4 is the processing of scenario $\mathcal{S}_0$ for a fragment of an NFA (the initial configuration of $\mathcal{Q}$), where A is the initial state. Notice that, since the ε-singularity is the first being processed, $\mathcal{Q}$ equals the input NFA $\mathcal{N}$, hence the extensions of the states in $\mathcal{Q}$ are all singletons involving the corresponding NFA state. Highlighted in the first configuration of $\mathcal{Q}$ displayed on the left side of Figure 4 are the states in $\overline{Q}$ (line 7), where the dashed ones are the unsafe states (cf. the UNSAFE auxiliary function in Section 4.3.1), namely C, D, and F. Then, the extension of the initial state is enlarged by the set of NFA states involved in $\overline{Q}$ (line 8), namely $\{1, 2, 3, 5\}$ (cf. the ENLARGE auxiliary procedure in Section 4.3.2), while the ε-transitions exiting the initial state are removed (line 9), which brings $\mathcal{Q}$ to the second configuration in Figure 4. The processing in lines 10–12 causes the insertion of the transitions $\langle A, b, G\rangle$ and $\langle A, b, H\rangle$, while lines 13–16 provoke the removal of the transition $\langle B, b, C\rangle$, which brings $\mathcal{Q}$ to the third configuration in Figure 4. Eventually, the processing in lines 17 and 18 brings $\mathcal{Q}$ to the last configuration (right side of Figure 4), where the singularity (F, c) has been removed (owing to the removal of F), which terminates the processing of scenario $\mathcal{S}_0$.

Scenario $\mathcal{S}_1$ (lines 22–28) occurs when no ℓ-transition exits q. Hence, a transition mapping (q, ℓ) to a state q' such that $\|q'\| = \overline{N}$ needs to be created. Two cases are possible: either q' exists or it does not. If q' exists already, then a transition $\langle q, \ell, q'\rangle$ is inserted (lines 23 and 24). Since the insertion of a new transition may shorten the level of the target state and possibly of other connected states, the auxiliary procedure LEVEL is called (Algorithm 6, Section 4.3.4) for the adjustment of the relevant levels. If, instead, q' does not exist, then it is created with extension $\|q'\| = \overline{N}$ by means of the NEW auxiliary function (Algorithm 5, Section 4.3.3) in line 26, which also generates the relevant singularities for the newly-created state. Moreover, a new transition $\langle q, \ell, q'\rangle$ is inserted and the level of q' is assigned (line 27).

Scenario $\mathcal{S}_2$ (lines 29–43) occurs when an ℓ-transition exiting q exists already. This scenario is reminiscent of scenario $\mathcal{S}_0$ and, hence, the repair actions are somewhat similar. The conditions in lines 29 and 30 avoid processing a singularity

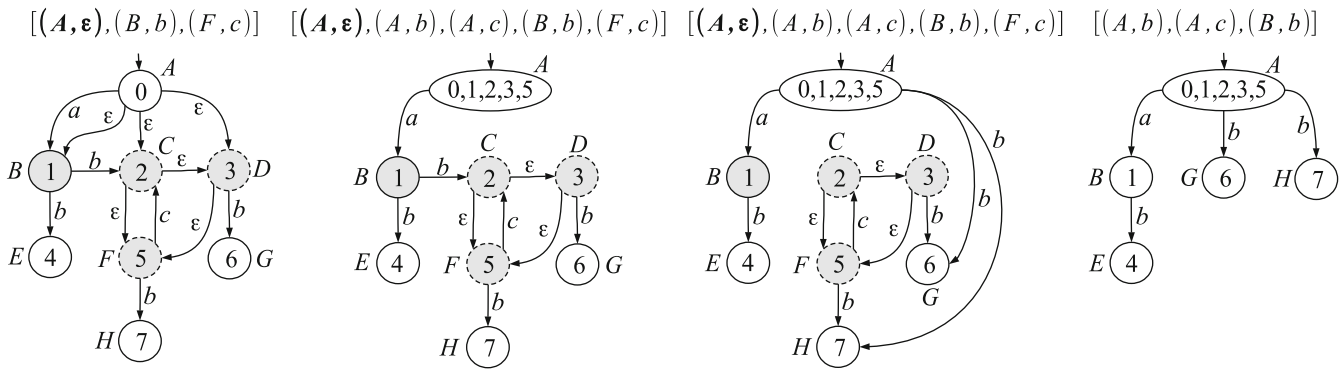

FIGURE 4 Processing of scenario $\mathcal{S}_0$ by QUICK SUBSET CONSTRUCTION. On top of each intermediate graph is the content of the singularity list, where the head (in bold) is the ε-singularity under processing (note how new singularities are generated). The states in gray are those in $\overline{Q}$ (namely, the ε-closure of q_0, line 7), where the unsafe states (those in $\bar{U}$) are dashed.

that does not change $\mathcal{Q}$, thereby enabling the repair actions only if the transition mapping (q, ℓ) actually needs adjustment. If so, first the sets $\overline{Q}$ and $\bar{U}$ are computed (line 31). Then a state q' is created by means of the auxiliary function NEW (line 32), where $\|q'\| = \overline{N}$ (with $\overline{N}$ being the ℓ-mapping of $\|q\|$ generated in line 21). Then, the ℓ-transitions exiting q are removed (line 33). Next, for each transition $\langle u, \ell', q'' \rangle$ where u is unsafe, $\ell \neq \varepsilon$, and q'' is not among the unsafe states, a transition $\langle q', \ell', q'' \rangle$ is inserted (lines 34–36). Like in scenario $\mathcal{S}_0$, these new transitions are meant to prevent $\mathcal{Q}$ from becoming disconnected owing to the subsequent removal of transitions in lines 37–40, where for each transition $\langle q'', \ell', u \rangle$ removed, with $\ell' \neq \varepsilon$, a singularity (q'', ℓ') is created. Then, all unsafe states and relevant transitions are removed (line 41). Afterwards, in line 42, a new transition $\langle q, \ell, q' \rangle$ is inserted into $\mathcal{Q}$, where q' is the new state created in line 32. The insertion of the new transition requires the setting of the level of q' as well as the possible update of the level of other states by means of the auxiliary procedure LEVEL. In case another existing state q'' turns out to have the same extension as q', in order to avoid duplication of states in $\mathcal{Q}$, q' and q'' are merged into a single state (line 43) by means of the UNIFY procedure (Algorithm 7, Section 4.3.5).

Example 8 (Scenario $\mathcal{S}_2$). Shown in Figure 5 is the processing of scenario $\mathcal{S}_2$ for a fragment of an intermediate configuration of $\mathcal{Q}$, where the relevant singularity is (C, a), with C being exited by two a-transitions. Highlighted in the configuration of $\mathcal{Q}$ in the left side of Figure 5 are the states in $\overline{Q}$ (line 31), where the unsafe states are dashed, namely E and G. Then, based on lines 32 and 33, assuming $\overline{N} = \{3, 4, 5, 6, 7, 8\}$ in line 21, a new state I is generated, with $\|I\| = \overline{N}$, and the transitions exiting C are removed (second configuration of $\mathcal{Q}$ in Figure 5). The processing in lines 34–36 causes the insertion of the transition $\langle I, c, H \rangle$, while lines 37–40 provoke the removal of the transition $\langle F, a, E \rangle$, which brings $\mathcal{Q}$ to the third configuration. Eventually, the processing in lines 41–43 brings $\mathcal{Q}$ to the last configuration in Figure 5, which terminates the processing of scenario $\mathcal{S}_2$.

Example 9 (QUICK SUBSET CONSTRUCTION). We now revisit the determinization of the NFA depicted on the left side of Figure 1 using QUICK SUBSET CONSTRUCTION, which has been carried out already by SUBSET CONSTRUCTION in Example 5 (cf. Figure 2). Shown in Figure 6 are the various configurations of $\mathcal{Q}$, which correspond to the processing of the singularities involved. Note that on top of each configuration of $\mathcal{Q}$ is the relevant content of the singularity list, where the singularity in first position (the head of the list, the first singularity to be processed) is in bold. On the left side of Figure 6 is the initial configuration of $\mathcal{Q}$ (the input NFA), along with the initial singularities, namely (C, a), since C is exited by two a-transitions, and (D, a), as there is a transition exiting D that enters the state D' which is exited by an ε-transition (cf. the rules for the initial singularities in Section 4.1). The first singularity (C, a) leads to scenario $\mathcal{S}_2$ which, once processed, brings $\mathcal{Q}$ to the second configuration, with two new singularities (relevant to E) being generated. The processing of the singularity (D, a) leads to scenario $\mathcal{S}_1$, which inserts the new transition $\langle D, a, E \rangle$ (third configuration in Figure 6). Processing the singularity (E, a) leads again to scenario $\mathcal{S}_1$, which inserts the new auto transition $\langle E, a, E \rangle$ (fourth configuration). Eventually, the singularity (E, b) has no effect because none of

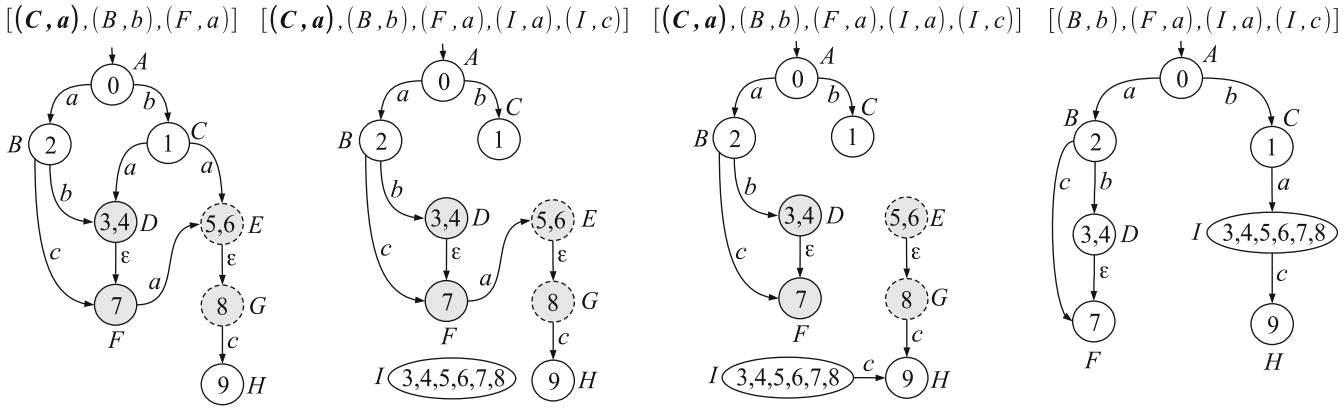

FIGURE 5 Processing of scenario $\mathcal{S}_2$ by QUICK SUBSET CONSTRUCTION. On top of each intermediate graph is the content of the singularity list, where the head (in bold) is the singularity under processing (note how new singularities are generated). The states in gray are those in $\overline{Q}$ (namely, the ℓ-mapping of q, line 31), where the unsafe states (those in $\bar{U}$) are dashed.

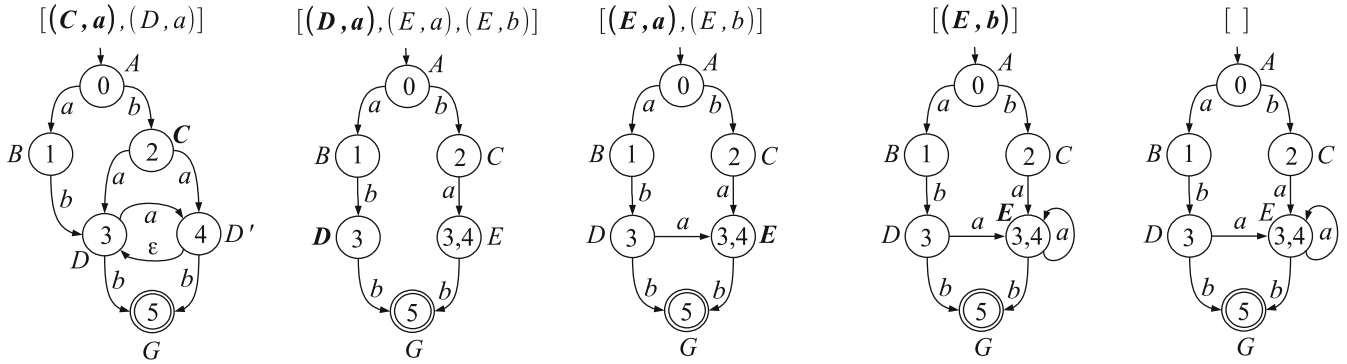

FIGURE 6 Transformation by QUICK SUBSET CONSTRUCTION of an NFA into its *SC*-equivalent DFA (cf. Figure 1). On top of each intermediate automaton is the content of the singularity list, where the head (in bold) is the singularity under processing. The algorithm terminates when the singularity list is empty.

Algorithm 3. *Unsafe* (auxiliary function)

1: **function** UNSAFE($(q, \ell), \bar{q}) \rightarrow$ *flag*
2: **input:** (q, ℓ): a singularity in $\mathcal{Q}$,
3: $\bar{q}$: a state in Q
4: **output:** *flag*: a Boolean value indicating whether $\bar{q}$ is unsafe
5: **begin**
6: *flag* $\leftarrow \bar{q} \neq q_0$ **and** there is no transition $\langle q^*, \ell^*, \bar{q}\rangle$ in δ_q such that $(q^*, \ell^*) \neq (q, \ell)$ and $\lambda(q^*) \leq \lambda(q)$
7: **end function**

the conditions in lines 29 and 30 is fulfilled, thereby bringing $\mathcal{Q}$ to the final configuration where the singularity list is empty, which terminates the execution of the algorithm. As expected, the resulting DFA (right side of Figure 6) is identical to the DFA generated from scratch by SUBSET CONSTRUCTION in Example 5 (cf. Figure 2). In other words, QUICK SUBSET CONSTRUCTION has progressively transformed the NFA into its SC-equivalent DFA.

4.3 | Auxiliaries

This section describes the five auxiliary functions/procedures exploited by QUICK SUBSET CONSTRUCTION, namely UNSAFE, ENLARGE, NEW, LEVEL, and UNIFY.

4.3.1 | UNSAFE

The pseudocode of the UNSAFE Boolean function is listed in Algorithm 3 (lines 1–7). It takes as input a singularity (q, ℓ) and a state $\overline{q}$ of $\mathcal{Q}$. It generates as output a *flag* indicating whether $\overline{q}$ is unsafe or not. The state $\overline{q}$ is unsafe if the removal of an ℓ-transition exiting q *might* cause the disconnection of $\overline{q}$ from the initial state q_0. The condition of unsafety is expressed in line 6, namely, when $\overline{q}$ is not the initial state and there is no transition $\langle q^*, \ell^*, \overline{q}\rangle$ either not exiting q or exiting q with a different label, such that the level of q^* is not greater than the level of q. If so, since the level may increase from one state to a successive one, every parent state of $\overline{q}$ might be a state that is reachable from $\overline{q}$ and, hence, removing the ℓ-transition exiting q may result in the disconnection of $\overline{q}$ and possibly of other states that are reachable from $\overline{q}$.

Example 10 (UNSAFE). Shown in Figure 7 is a fragment of $\mathcal{Q}$ when the UNSAFE function is called, where $(q, \ell) = (C, a)$. Within the set of gray states, the Boolean *flag* is true when $\overline{q}$ is a dashed state (E, F, H, I, and J), while it is false when the state is plain (D and G). For instance, the state E is unsafe because the condition

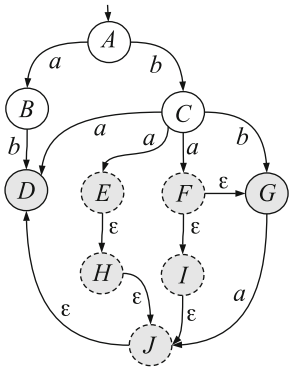

FIGURE 7 Portion of a configuration of $\mathcal{Q}$ when the auxiliary function UNSAFE is called, where $(q, \ell) = (C, a)$.

Algorithm 4. *Enlarge* (auxiliary procedure)

```
1:  procedure ENLARGE(q, N̄)
2:      input: q: a state of Q,
3:             N̄: a set of states of the NFA 𝒩
4:      side effects: the extension of q is enlarged by N̄ and new singularities are possibly generated for q
5:  begin
6:      if N̄ ⊈ ‖q‖ then
7:          for all ℓ ∈ Σ such that there is a state n ∈ (N̄ \ ‖q‖) that is exited by an ℓ-transition in 𝒩 do
8:              if a singularity (q, ℓ) does not exist then generate a new singularity (q, ℓ) end if
9:          end for
10:         ‖q‖ ← ‖q‖ ∪ N̄
11:     end if
12: end procedure
```

in line 6 is not fulfilled, specifically, because E is not the initial state, nor is there a transition $\langle q^*, \ell, E\rangle$ such that $(q^*, \ell) \neq (C, a)$ and $\lambda(q^*) \leq \lambda(C)$. By contrast, the state D is not unsafe owing to the transition $\langle B, b, D\rangle$ which fulfills the condition in line 6 (in fact, $\lambda(B) = \lambda(C) = 1$). This means that the removal of the transition $\langle C, a, D\rangle$ will leave D still reachable from the initial state thanks to the connection with B.[**]

4.3.2 | ENLARGE

The pseudocode of the ENLARGE procedure is listed in Algorithm 4 (lines 1–12). It takes as input a state q of $\mathcal{Q}$ and a set $\overline{N}$ of states of $\mathcal{N}$ (the NFA given in input to QUICK SUBSET CONSTRUCTION). As a side effect, the extension of q is expanded by $\overline{N}$, possibly with new singularities being created for q. The condition in line 6 prevents that expansion from being immaterial. Thus, if there is at least one state in $\overline{N}$ that is not included in the current extension of q, then new singularities for q are possibly generated in lines 7–9, specifically, a singularity (q, ℓ) for each label ℓ marking a transition exiting a state in $\mathcal{N}$ that is in $\overline{N}$ but not in the current extension of q, thereby allowing for the subsequent adjustment of the transition function relevant to these labels. Eventually, the extension of q is enlarged by $\overline{N}$ (line 10).

[**] A state qualified as unsafe may be still reachable from the initial state after the removal of a transition (q, ℓ). For instance, although being qualified as unsafe, state J is still connected with the initial state when all the a-transitions exiting C are removed, owing to its connection with G. In other words, an unsafe state *might* be disconnected from the initial state, but not for sure. By contrast, if a state is not unsafe, it will remain connected with the initial state for sure.

Algorithm 5. *New* (auxiliary function)

1: **function** NEW($\bar{N}$) $\rightarrow q$
2: **input:** $\bar{N}$: a set of states of the NFA $\mathcal{N}$
3: **side effects:** a new state q is generated in $\mathcal{Q}$, where $\|q\| = \bar{N}$, along with relevant singularities
4: **output:** q: the newly-created state
5: **begin**
6: Insert a new state q into Q, where $\|q\| = \emptyset$
7: ENLARGE($q, \bar{N}$)
8: **end function**

Algorithm 6. *Level* (auxiliary procedure)

1: **procedure** LEVEL(Λ)
2: **input:** $\Lambda = [(q, \lambda)]$: a singleton list, where $q \in Q$ and λ is the actual level of q
3: **side effects:** the level of q and of other states reachable from q is possibly updated
4: **begin**
5: **repeat**
6: Remove the head $(\bar{q}, \bar{\lambda})$ from the list Λ
7: **if** $\lambda(\bar{q})$ is unassigned **or** $\lambda(\bar{q}) > \bar{\lambda}$ **then**
8: $\lambda(\bar{q}) \leftarrow \bar{\lambda}$
9: **for all** transition $\langle \bar{q}, \ell, q' \rangle \in \delta_q$ **do**
10: Append $(q', \bar{\lambda} + 1)$ to Λ
11: **end for**
12: **end if**
13: **until** Λ is empty
14: **end procedure**

4.3.3 | NEW

The pseudocode of the NEW function is listed in Algorithm 5 (lines 1–8). It takes as input a set of states of $\mathcal{N}$ (the NFA given in input to QUICK SUBSET CONSTRUCTION) and creates a new state q in $\mathcal{Q}$ having extension $\overline{N}$. First, q is created as an empty state (line 6). Then, the (empty) extension of q is extended by $\overline{N}$ by calling the ENLARGE procedure, hence, possibly generating new singularities for q also (cf. Algorithm 4).

4.3.4 | LEVEL

The pseudocode of the LEVEL procedure is listed in Algorithm 6 (lines 1–14). It takes as input a singleton list $\Lambda = [(q, \lambda)]$, where q is a state of $\mathcal{Q}$, possibly with unassigned level, and λ is the actual level of q. As a side effect, the level of q, as well as of other states that are reachable from q, is possibly updated. This is necessary whenever a new transition inserted into $\mathcal{Q}$ may shorten the level of some states (lines 11, 24, and 42 of QUICK SUBSET CONSTRUCTION). A loop is performed in lines 5–13, where the first pair $(\overline{q}, \overline{\lambda})$ in Λ is considered (line 6). If the level of $\overline{q}$ is not yet assigned or the current level of $\overline{q}$ is greater than the actual level $\overline{\lambda}$, then the level of $\overline{q}$ is assigned with $\overline{\lambda}$ (line 8). Since this assignment may cause a change in the level of the successive states of $\overline{q}$, for every state q' that is entered by a transition exiting $\overline{q}$, a new pair $(q', \overline{\lambda} + 1)$ is appended to Λ in order to subsequently propagate the change of levels to the states that are reachable from $\overline{q}$ (lines 9–11). The loop terminates when Λ becomes empty, that is, when all relevant levels have been updated (line 13).

Example 11 (LEVEL). Shown on the left side of Figure 8 is a configuration of the FA $\mathcal{Q}$ being manipulated by QUICK SUBSET CONSTRUCTION, where each state is marked with the relevant level (whereas transition labels

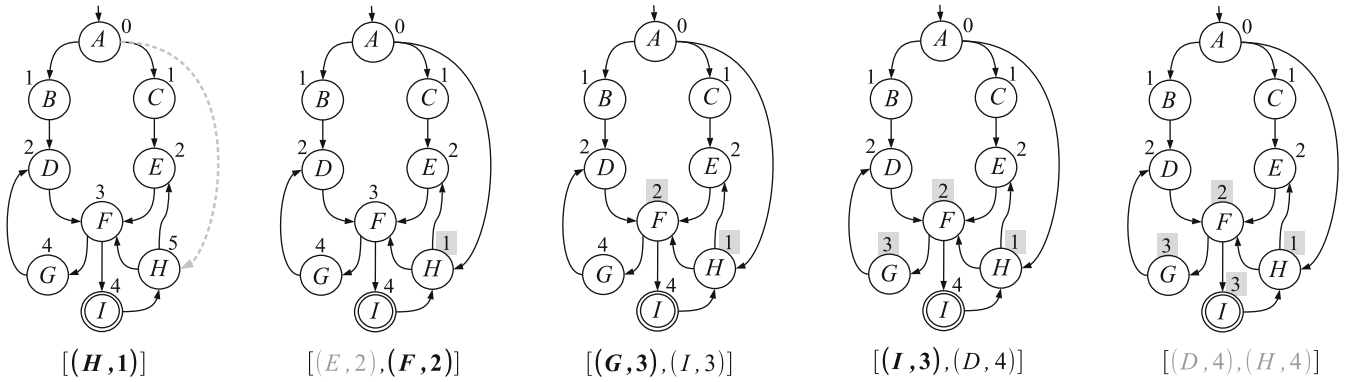

FIGURE 8 Processing by the auxiliary procedure LEVEL after the insertion of a transition from A to H performed by QUICK SUBSET CONSTRUCTION, which causes a reduction in the levels of the states F, G, H, and I (transition labels are omitted). Displayed below each graph configuration is the instance of Λ, where the processing of pairs in gray has no effect on levels.

are omitted). Let assume now that a new transition from A to H is inserted into Q (dashed gray arrow). This insertion causes a call to procedure LEVEL with input parameter $\Lambda = [(H, 1)]$, as expressed below the graph. According to lines 7–12 of LEVEL, the processing of the pair $(H, 1)$ changes the level of H to 1 and appends the new pairs $(E, 2)$ and $(F, 2)$ to the list Λ (second graph). The processing of $(E, 2)$ has no effect on the level, thereby leaving the graph unchanged. Instead, the subsequent processing of $(F, 2)$ changes the level of F to 2, while appending the pairs $(G, 3)$ and $(I, 3)$ (third graph). Next, processing $(G, 3)$ changes the level of G to 3 and appends the pair $(D, 4)$ (fourth graph). Processing $(I, 3)$ changes the distance of I to 3 and appends the pair $(H, 4)$ (last graph on the right). Eventually, the processing of both $(D, 4)$ and $(H, 4)$ has no effect on the levels, thereby leaving the graph unchanged. Since now Λ is empty, the procedure LEVEL terminates. As expected, the levels of the states F, G, H, and I are reduced in accordance with the new topology of the graph after the insertion of the transition.

4.3.5 | UNIFY

The pseudocode of the UNIFY procedure is listed in Algorithm 7 (lines 1–14). It takes as input a newly-created state q' in Q (cf. scenario S_2 in QUICK SUBSET CONSTRUCTION) and another state q'' in Q that has the same extension as q'. In order to avoid duplication of states, all the transitions and singularities relevant to q' are inherited by q'', whereas the duplicated state q' is removed. First, the unique transition entering q' is redirected towards q'' (line 6). Since this redirection may cause a change in the level of d'', as well as of other states reached by q'', level relocation is performed by the LEVEL procedure. Then, all transitions exiting q' are moved to q'' (lines 7 and 8). Even in this case, since this redirection may cause changes in the level of the states entered by these transitions, LEVEL needs to be called. Afterwards, every singularity relevant to q' is transformed into a singularity relevant to q'' (lines 10–12). The (isolated) duplicated state q' is eventually removed from Q (line 13).

Example 12 (UNIFY). Shown on the left side of Figure 9 is a configuration of Q when the UNIFY procedure is called by QUICK SUBSET CONSTRUCTION (line 43) with input parameters $q' = F$ (to be removed) and $q'' = G$ (to be preserved). The new configuration of Q resulting from the unification of F and G is displayed on the right side of the figure, where F has been removed, while all its entering/exiting transitions (along with relevant singularities, omitted in the figure) have been moved to G. Note how G is eventually exited by two b-transitions, which in principle would require generating a singularity (G, b). More generally, redirecting towards q'' the transitions exiting q' may cause the transition function of q'' to become nondeterministic, which therefore requires the generation of new singularities for q''. In fact, however, according to line 32 of QUICK SUBSET CONSTRUCTION, the creation of the new state q' by NEW (cf. Section 4.3.3) invariably creates all the singularities of q' by means of the ENLARGE auxiliary procedure (cf. Section 4.3.2). Consequently, when q'' inherits the singularities of q', it also inherits the singularities that should be crated owing to the possible new nondeterminism stemming in q'. In other words, there is no need for the creation of new singularities for q''.

Algorithm 7. *Unify* (auxiliary procedure)

1: **procedure** UNIFY(q', q'')
2: **input:** q': a newly-created state in Q (scenario S_2),
3: q'': an existing state in Q where $\|q''\| = \|q'\|$,
4: **side effects:** all transitions and singularities relevant to q' are inherited by q'', whereas q' is removed
5: **begin**
6: Replace the (single) transition $\langle q, \ell, q' \rangle$ entering q' with $\langle q, \ell, q'' \rangle$, LEVEL($[(q'', \lambda(q) + 1)]$)
7: **for all** transition $t = \langle q', \ell', q^* \rangle \in \delta_q$ **do**
8: Replace t with $\langle q'', \ell', q^* \rangle$, LEVEL($[(q^*, \lambda(q'') + 1)]$)
9: **end for**
10: **for all** singularity (q', ℓ') **do**
11: Replace (q', ℓ') with (q'', ℓ')
12: **end for**
13: Remove q' from Q
14: **end procedure**

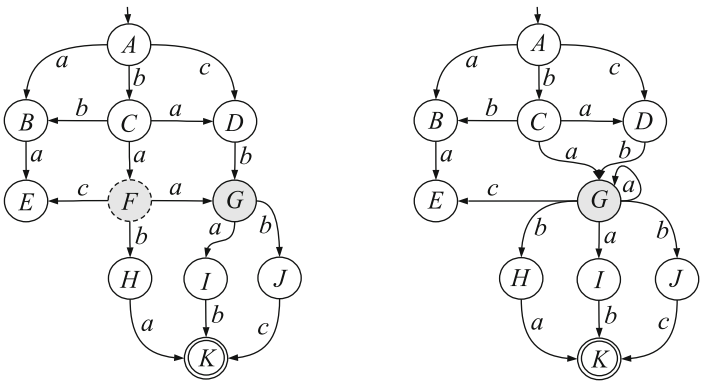

FIGURE 9 Processing by the auxiliary procedure UNIFY applied to $q' = F$ and $q'' = G$, where $\|F\| = \|G\|$.

5 | ALGORITHM EQUIVALENCE AND DFA IDENTITY

Massive experimentation has convincingly shown that QUICK SUBSET CONSTRUCTION generates a DFA that is *SC*-equivalent to the input NFA, in other words, that QUICK SUBSET CONSTRUCTION is equivalent to SUBSET CONSTRUCTION, inasmuch it generates a DFA that is identical to the DFA generated by SUBSET CONSTRUCTION. Empirical evidence, however, cannot prove the equivalence of the two algorithms conclusively. This is why a proof of correctness of QUICK SUBSET CONSTRUCTION is provided hereafter (Theorem 1), which shows that the DFA resulting from the repair actions performed on the NFA by this algorithm is identical to the DFA generated from scratch by SUBSET CONSTRUCTION. The proof makes use of the notion of a *path* of the algorithm.

Definition 2 (execution point and *path*). Let $\mathcal{N}$ be an NFA applied to QUICK SUBSET CONSTRUCTION and let Q be the FA being processed by QUICK SUBSET CONSTRUCTION. Let Q_i, $i \geq 0$, be a configuration of Q when a singularity is considered (in either line 6 or 20), and let σ_i be the configuration of the singularity list when such a singularity is considered. The pair (Q_i, σ_i) is an execution point of QUICK SUBSET CONSTRUCTION. The sequence $[(Q_0, \sigma_0), (Q_1, \sigma_1), \ldots]$ of execution points relevant to the processing of all the singularities is the *path* of QUICK SUBSET CONSTRUCTION when applied to $\mathcal{N}$.

Theorem 1. *Let $\mathcal{N} = (\Sigma, N, \delta_n, n_0, F_n)$ be an NFA, where every state is reachable from the initial state, a final state is reachable from every state, and there is no (auto) ε-transition $\langle n, \varepsilon, n \rangle$.*[††] *Let $\mathcal{D} = (\Sigma, D, \delta_d, d_0, F_d)$ be the*

[††]These assumptions do not affect the generality of $\mathcal{N}$, as the removal of the unwanted states and transitions does not change the regular language of $\mathcal{N}$.

DFA generated by the application of Subset Construction to $\mathcal{N}$ (namely the DFA that is SC-equivalent to $\mathcal{N}$), and let $\mathcal{Q}' = (\Sigma, Q, \delta_q, q_0, F_q)$ be the FA resulting from the application of Quick Subset Construction on $\mathcal{N}$. We have $\mathcal{Q}' = \mathcal{D}$.[‡‡]

Proof. The proof of the theorem is grounded on Definitions 3,4, and Lemmas 1–7. ∎

Definition 3 (completable transition). Let $(\mathcal{Q}_i, \sigma_i)$ be an execution point of QUICK SUBSET CONSTRUCTION and let $\langle q, \ell, q' \rangle$, $\ell \neq \varepsilon$, be a transition in $\mathcal{Q}_i$. If $\|q'\| = \ell$-mapping$(\|q\|, \mathcal{N})$, then $\langle q, \ell, q' \rangle$ is *complete*, otherwise the transition is *incomplete*. An incomplete transition $\langle q, \ell, q' \rangle$ in $\mathcal{Q}_i$ is *completable* if and only if σ_i includes a singularity (q, ℓ).

Definition 4 (*viability*). Let $(\mathcal{Q}_i, \sigma_i)$ be an execution point of QUICK SUBSET CONSTRUCTION. If every incomplete transition in $\mathcal{Q}_i$ is completable, then the execution point $(\mathcal{Q}_i, \sigma_i)$ is *viable*.

Lemma 1. *Quick Subset Construction terminates.*

Proof. By contradiction, assume that QUICK SUBSET CONSTRUCTION may not terminate. Since no recursive call is performed, this requires that a loop is executed endlessly. With regard to the auxiliary functions/procedures, only LEVEL includes a loop to be considered for possible nontermination, as the input (singleton) list Λ is possibly extended within the loop whose termination requires the emptiness of Λ. Assume that LEVEL does not terminate. Since each new pair $(q', \overline{\lambda} + 1)$ appended to Λ in line 10 of LEVEL is such that q' is a successive state of q (that is, a state reachable from q, the state relevant to the initial pair in Λ), sooner or later, the same state q' needs to be considered again in line 6, namely $\overline{q}$. However, $\overline{\lambda}$ cannot be less than the current level of $\overline{q}$, as the level $\overline{\lambda}$ continues growing in the processing, in other words, the loop in LEVEL must terminate, a contradiction.

Hence, the only mode in which QUICK SUBSET CONSTRUCTION may not terminate is by traversing an endless path, namely $\mathcal{P} = [(\mathcal{Q}_0, \sigma_0), (\mathcal{Q}_1, \sigma_1), \ \dots \ (\mathcal{Q}_k, \sigma_k), \ \dots \ ]$. Let $(\mathcal{Q}_i, \sigma_i)$, $i \geq 0$, be an execution point in $\mathcal{P}$. Since both the set of states in $\mathcal{Q}_i$ and the alphabet Σ are finite, σ_i is finite also, being an ordered set of pairs (q, ℓ), where q is a state in $\mathcal{Q}_i$ and ℓ a symbol in Σ. Besides, since the set of transitions in $\mathcal{Q}_i$ is finite, the set of possible execution points is certainly finite. Hence, a necessary condition for the path $\mathcal{P}$ to be infinite is to reach an execution point at least twice, that is $\mathcal{P} = [\ \dots\ , (\mathcal{Q}_i, \sigma_i), \ \dots\ , (\mathcal{Q}_j, \sigma_j), \ \dots\ ]$ where $(\mathcal{Q}_i, \sigma_i) = (\mathcal{Q}_j, \sigma_j)$. Notice that, since the processing of QUICK SUBSET CONSTRUCTION is deterministic, if an execution point is repeated, then the algorithm loops endlessly on this point (and all the other points within the loop). Hence, since the set of execution points is finite, to prove that $\mathcal{P}$ is finite (that is, QUICK SUBSET CONSTRUCTION terminates), it suffices to show that every execution point cannot be reached more than once in the path $\mathcal{P}$. To this end, we consider each of the three scenarios when QUICK SUBSET CONSTRUCTION processes a singularity (q, ℓ).

In scenario $\mathcal{S}_0$, the singularity being processed is (q_0, ε), while the execution point is $(\mathcal{Q}_0, \sigma_0)$, that is, the first one in the path. Since the singularity (q_0, ε) cannot be generated again subsequently, $(\mathcal{Q}_0, \sigma_0)$ cannot be reached again.

In scenario $\mathcal{S}_1$, assuming an execution point $(\mathcal{Q}_i, \sigma_i)$, the singularity (q, ℓ) to be processed is such that there is no ℓ-transition exiting q; thus, a transition $\langle q, \ell, q' \rangle$ is generated, with q' being possibly created. Notice that, in subsequent processing, q' cannot be unsafe, as a subsequent singularity $(\overline{q}, \overline{\ell})$ is such that $\lambda(q) \leq \lambda(\overline{q})$. Hence, the transition $\langle q, \ell, q' \rangle$ cannot be deleted subsequently to possibly repeat the execution point $(\mathcal{Q}_i, \sigma_i)$. Even a subsequent unification of another state with q by UNIFY in a new execution point $(\mathcal{Q}_j, \sigma_j)$, $i < j$, cannot repeat $(\mathcal{Q}_i, \sigma_i)$ because q will be exited by an ℓ-transition, a condition that makes $\mathcal{Q}_j$ different from $\mathcal{Q}_i$.

In scenario $\mathcal{S}_2$, similarly to scenario $\mathcal{S}_1$, assuming an execution point $(\mathcal{Q}_i, \sigma_i)$, the generation of the new transition $\langle q, \ell, q' \rangle$ makes q' safe in subsequent processing, so that this transition cannot be removed afterwards. Hence, even a subsequent unification of another state with q by UNIFY in a new execution point $(\mathcal{Q}_j, \sigma_j)$, $i < j$, cannot repeat $(\mathcal{Q}_i, \sigma_i)$ because either there will be a single ℓ-transition exiting q (rather than several ones) or $\|q'\|$ will differ from the extension of the target state of the (single) transition exiting q in $\mathcal{Q}_i$; in other

[‡‡]In other words, when the same NFA $\mathcal{N}$ is given in input to both SUBSET CONSTRUCTION and QUICK SUBSET CONSTRUCTION, the DFA generated from scratch by SUBSET CONSTRUCTION is identical to the DFA obtained by means of the repair actions performed by QUICK SUBSET CONSTRUCTION.

words, $(\mathcal{Q}_i, \sigma_i)$ cannot be reached again. In conclusion, assuming that the algorithm does not terminate leads to a contradiction; hence, QUICK SUBSET CONSTRUCTION does terminate. ∎

Lemma 2. *Every state in $\mathcal{Q}'$ is reachable from the initial state.*

Proof. The proof is by induction on the (finite) path of QUICK SUBSET CONSTRUCTION, $[(\mathcal{Q}_0, \sigma_0), (\mathcal{Q}_1, \sigma_1), \ \dots \ , (\mathcal{Q}_m, \sigma_m)]$.

(*Basis*) *Every state in $\mathcal{Q}_0$ is reachable from the initial state.* Since $\mathcal{Q}_0 = \mathcal{N}$, this is true by assumption.

(*Induction*) *If every state in $\mathcal{Q}_i$, $i \geq 0$, is reachable from the initial state, then every state in $\mathcal{Q}_{i+1}$ is reachable from the initial state.* Note that the only way to have a disconnection from the initial state q_0 is by removal of transitions, which may hold in scenarios $\mathcal{S}_0$ and $\mathcal{S}_2$ only. In scenario $\mathcal{S}_0$, all the ε-transitions exiting q_0 are removed (line 9). Among the target states of the ε-transitions removed, the states that are not unsafe keep being reached from q_0. Apparently, instead, all the (unsafe) states in $\bar{U}$ (including the target states of the ε-transitions removed), might become disconnected from q_0. Moreover, since all states in $\bar{U}$ are removed along with their entering/exiting transitions (line 17), the target states of these exiting transitions might be disconnected also. However, the transitions $\langle q_0, \ell, q\rangle$ inserted in line 11 make these target states still reachable from q_0. Finally, the transitions $\langle q, \ell, u\rangle$ removed in line 14 are irrelevant to the disconnection of the target (unsafe) states $u \in \bar{U}$ because these states are removed and, as shown above, the removal of their exiting transitions does not cause any disconnection from q_0. The actions performed in scenario $\mathcal{S}_0$ to save the connection with q_0 are somewhat replicated in scenario $\mathcal{S}_2$, with the exception of the generation of a new transition $\langle q, \ell, q'\rangle$ (line 42). In line 33, all the ℓ-transitions exiting q are removed. Among the target states of the ℓ-transitions removed, the states that are not unsafe keep being reached from q and, hence, from q_0. All the (unsafe) states in $\bar{U}$ (including the target states of the ℓ-transitions removed), instead, might become disconnected from q, and, consequently, from q_0. Besides, since all states in $\bar{U}$ are removed along with their entering/exiting transitions (line 41), the target states of these exiting transitions might be disconnected also. However, the transitions $\langle q', \ell', q''\rangle$ inserted in line 35 make these target states still reachable from q and, hence, from q_0. Finally, the transitions $\langle q'', \ell', u\rangle$ removed in line 38 are irrelevant to the disconnection of the target (unsafe) states $u \in \bar{U}$ because these states are removed and, as shown above, the removal of their exiting transitions does not cause any disconnection from q_0. Finally, the possible unification of the new state q' with an existing state q'' in line 43 by means of UNIFY is apparently another source of possible disconnection. Based on UNIFY, the transition entering q' as well as all the transitions exiting q' are moved to q'' (lines 6 and 8, respectively), whereas q' is eventually removed (line 13). However, since q'' is reachable from q_0, all the target states of the transitions exiting q' and inherited by q'' continue being reachable from q_0. ∎

Lemma 3. *Every transition in $\mathcal{Q}'$ is complete.*

Proof. Let $\mathcal{P} = [(\mathcal{Q}_0, \sigma_0), (\mathcal{Q}_1, \sigma_1), \ \dots \ , (\mathcal{Q}_m, \sigma_m)]$ be a path of QUICK SUBSET CONSTRUCTION, which, according to Lemma 1, is finite, in other words, $\mathcal{Q}_m = \mathcal{Q}'$ and σ_m is empty. We show by induction that every execution point $(\mathcal{Q}_i, \sigma_i)$, $i \in [0\,..\,m]$, is viable (cf. Definition 4).

(*Basis*) *Every incomplete transition in $\mathcal{Q}_0$ is completable.* In fact, $\mathcal{Q}_0 = \mathcal{N}$ and σ_0 is the list of initial singularities (cf. Section 4.1), which are relevant to the points of nondeterminism in $\mathcal{N}$. Apart from the singularity (q_0, ε), which is however irrelevant to the notion of completeness as it refers to ε-transitions exiting q_0, each other singularity (q, ℓ) is associated with the ℓ-transitions exiting q, which are therefore incomplete. In fact, either several ℓ-transitions exit q or we have $\langle q, \ell, q'\rangle$ where q' is exited by an ε-transition. In either case, $\|q'\| \neq \ell$-mapping$(\|q\|, \mathcal{N})$, that is, these transitions are incomplete. Based on Definition 3, however, these incomplete transitions are completable owing to the initial singularities (q, ℓ).

(*Induction*) *If every incomplete transition in $\mathcal{Q}_i$ is completable, $i \geq 0$, then every incomplete transition in $\mathcal{Q}_{i+1}$ is completable.* We have to show that all incomplete ℓ-transitions in $\mathcal{Q}_{i+1}$ are still completable once the relevant singularity (q, ℓ) has been processed in either scenario $\mathcal{S}_1$ or $\mathcal{S}_2$. In scenario $\mathcal{S}_1$, each of the two transitions generated in line 24 and 27, respectively, is complete. In scenario $\mathcal{S}_2$, the transition generated in line 42 is complete, while every incomplete transition generated in line 35 is completable, as the creation of q' in line 32 comes with the relevant singularities. If unification comes into play (line 43), then the inheritance of the singularities (q', ℓ) relevant to q' by q'' (the state that has same extension as q') makes each incomplete transition

(q'', ℓ) still completable. Hence, all incomplete transitions in $\mathcal{Q}_{i+1}$ are completable, in other words, $(\mathcal{Q}_{i+1}, \sigma_{i+1})$ is viable.

Eventually, since in $\mathcal{Q}' = \mathcal{Q}_m$ every incomplete transition is completable and σ_m is empty (no singularity exists), no transition is incomplete (otherwise a singularity would exist), in other words, every transition in $\mathcal{Q}'$ is complete. ∎

Lemma 4. *$\mathcal{Q}'$ is deterministic.*

Proof. This is a consequence of Lemma 3. By contradiction, assume that $\mathcal{Q}'$ is nondeterministic. As such, $\mathcal{Q}'$ either includes an ε-transition or a state q that is exited by several ℓ-transitions. But, on the one hand, no ε-transition may exist in $\mathcal{Q}'$ because scenario $\mathcal{S}_0$ removes all the ε-transitions exiting the initial state q_0, while all other ε-transitions in $\mathcal{N}$ are eliminated by the relevant initial singularities, after which no other ε-transition may be generated. On the other hand, if there are in $\mathcal{Q}'$ two ℓ-transitions $\langle q, \ell, q' \rangle$ and $\langle q, \ell, q'' \rangle$ exiting q, where $\|q'\| \neq \|q''\|$, then, owing to Lemma 3, $\|q'\| = \|q''\| = \ell$-mapping$(\|q\|, \mathcal{N})$, a contradiction. Hence, nondeterminism cannot hold in $\mathcal{Q}'$, in other words, $\mathcal{Q}'$ is deterministic. ∎

Lemma 5. *The extension of the initial state of $\mathcal{Q}'$ equals the extension of the initial state of $\mathcal{D}$.*

Proof. We have to show that $\|q_0\| = \|d_0\|$. Based on SUBSET CONSTRUCTION, $\|d_0\| = \varepsilon$-closure$(n_0, \mathcal{N})$, where n_0 is the initial state of $\mathcal{N}$. If n_0 is not exited by any ε-transition in $\mathcal{N}$, then $\|d_0\| = \{n_0\} = \|q_0\|$, as there is no ε-singularity (q_0, ε) in $\mathcal{Q}$, in other words, scenario $\mathcal{S}_0$ is not applicable. If, instead, n_0 is exited by at least one ε-transition, then there is the initial singularity (q_0, ε) in $\mathcal{Q}$, which, based on scenario $\mathcal{S}_0$, provokes in line 8 the enlargement of $\|q_0\|$ to $\overline{N} = \varepsilon$-closure$(n_0, \mathcal{N})$, as $\overline{Q}$ is the set of singleton states in $\mathcal{Q}$ corresponding to the states in ε-closure$(n_0, \mathcal{N})$. Since the extension of q_0 cannot be changed afterwords by QUICK SUBSET CONSTRUCTION, we have in any case $\|q_0\| = \|d_0\|$. ∎

Lemma 6. *The transition function of $\mathcal{Q}'$ equals the transition function of $\mathcal{D}$.*

Proof. Let $d \in D$ and $q \in Q$, where $\|d\| = \|q\|$. We first show that the transition function of d (set of transitions exiting d in $\mathcal{D}$) equals the transition function of q (set of transitions exiting q in $\mathcal{Q}'$).

(*Soundness*) *If* $\langle q, \ell, q' \rangle \in \delta_{\mathrm{q}}$, *then* $\langle d, \ell, d' \rangle \in \delta_{\mathrm{d}}$ *and* $\|d'\| = \|q'\|$. Notice that, if an ℓ-transition exits q in $\mathcal{Q}'$, then $\|q\|$ includes a state n that is exited by an ℓ-transition in $\mathcal{N}$. In fact, this is certainly true initially, when $\mathcal{Q} = \mathcal{N}$. Subsequently, the enlargement of $\|q_0\|$ in scenario $\mathcal{S}_0$ along with the generation of the transitions exiting q_0 (line 11), as well as the creation of the new state q' in scenario $\mathcal{S}_2$ along with the generation of the transitions exiting q' (line 35), preserve this property, which holds in scenario $\mathcal{S}_1$ also. Therefore, based on Lemma 3, $\|q'\| = \ell$-mapping$(\|q\|, \mathcal{N})$. On the other hand, since $\|d\| = \|q\|$, we have $n \in \|d\|$; hence, $\langle d, \ell, d' \rangle \in \delta_{\mathrm{d}}$, where $\|d'\| = \|q'\|$.

(*Completeness*) *If* $\langle d, \ell, d' \rangle \in \delta_{\mathrm{d}}$, *then* $\langle q, \ell, q' \rangle \in \delta_{\mathrm{q}}$ *and* $\|q'\| = \|d'\|$. If $\langle d, \ell, d' \rangle$ is also in $\mathcal{N}$, then $\|d\| = \{n\}$ and $\|d'\| = \{n'\}$. Thus, since the transition is deterministic already, no singularity (q, ℓ) is generated by QUICK SUBSET CONSTRUCTION; hence, $\langle q, \ell, q' \rangle \in \delta_{\mathrm{q}}$, where $\|q'\| = \|d'\| = \{n'\}$. If, instead, either $\|d\|$ is not a singleton or $\|d\| = \{n\}$ where ℓ-mapping$(\{n\}, \mathcal{N})$ is not a singleton (nondeterminism), then notice that, once q is generated, if a state n belonging to $\|q\|$ is exited by an ℓ-transition in $\mathcal{N}$, then either q is exited by an ℓ-transition or there is a singularity (q, ℓ). Hence, in any case, based on Lemma 3, $\langle q, \ell, q' \rangle \in \delta_{\mathrm{q}}$, where $\|q'\| = \|d'\|$.

Since the transition function of d equals the transition function of q, based on Lemma 5, it follows inductively that $\delta_{\mathrm{d}} = \delta_{\mathrm{q}}$. ∎

Lemma 7. *The set of states of $\mathcal{Q}'$ equals the set of states of $\mathcal{D}$.*

Proof. This is a consequence of Lemmas 5 and 6.

Based on the lemmas above, assuming that the set of final states is computed correctly by QUICK SUBSET CONSTRUCTION, the proof of Theorem 1 is eventually grounded on these facts: QUICK SUBSET CONSTRUCTION always terminates (Lemma 1), $\mathcal{Q}'$ is deterministic (Lemma 4), each state in $\mathcal{Q}'$ is reachable from the initial state (Lemma 2), the initial state of $\mathcal{Q}'$ equals the initial state of $\mathcal{D}$ (Lemma 5), the set of states in $\mathcal{Q}'$ equals the set of states in $\mathcal{D}$ (Lemma 7), the transition function of $\mathcal{Q}'$ equals the transition function of $\mathcal{D}$ (Lemma 6). ∎

6 | EXPERIMENTATION

Both SUBSET CONSTRUCTION and QUICK SUBSET CONSTRUCTION were implemented in a software framework by means of the C++ programming language, under the GNU/Linux 5.4.0-42-generic x86_64 (Ubuntu 18.04.5 LTS) operating system, on a machine with Intel(R) Xeon(R) Gold 6140M CPU (2.30GHz) and 128 GB of working memory.[§§] This framework was also required to generate pseudo-random test cases based on a variety of user-defined parameters, as well as to analyze the aggregate data generated by the relevant experiments. The software framework was exploited mainly to:

1. Verify the equivalence of QUICK SUBSET CONSTRUCTION and SUBSET CONSTRUCTION empirically;
2. Compare the performance of the two algorithms in terms of processing time;
3. Suggest the conditions under which QUICK SUBSET CONSTRUCTION may perform better than SUBSET CONSTRUCTION.

The main parameters considered for the pseudo-random generation of NFAs are:

- *Number of states*;
- *Branching factor*: the average number of transitions exiting a state;
- *Epsilon density*: the percentage of ε-transitions;
- *Height*: the number of strata composing a stratified NFA (cf. Section 6.1);
- *Determinism*: the percentage of contiguous strata with deterministic transition function in a stratified NFA (cf. Section 6.1);[¶¶]
- *Alphabet cardinality*: the number of labels in the alphabet;
- *Final density*: the percentage of final states.

Once implemented the framework, a phase of experimentation was conducted in order to test the correctness of QUICK SUBSET CONSTRUCTION empirically and to compare its performance with that of SUBSET CONSTRUCTION. A large number of experiments were carried out on various typologies of finite automata based on different parameter configurations.

6.1 | Preliminaries

A recurring conjecture of this article is that the *raison d'être* of QUICK SUBSET CONSTRUCTION lies in its ability to outperform SUBSET CONSTRUCTION when the NFA is large and the nondeterminism affecting the NFA can be removed by a limited number of repair actions (as compared to the size of the *SC*-equivalent DFA). Intuitively, these two conditions combined allow QUICK SUBSET CONSTRUCTION to operate more efficiently than SUBSET CONSTRUCTION does because only a (possibly tiny) fraction of the NFA is updated, thereby leaving a (possibly large) deterministic portion of the NFA untouched. As pointed out in Section 4.1, however, the number of repair actions actually performed (that is, the number of singularities processed) is not directly related to the quantity of nondeterminism in the NFA (that is, the number of initial singularities, cf. Definition 1). In other words, *limited quantity of nondeterminism in an NFA does not necessarily translate into limited processing by* QUICK SUBSET CONSTRUCTION: much depends on how the states are connected to one another. For example, if a singularity affects the initial state of the NFA, in the worst case, all the states of the NFA may be involved in the repair actions owing to the cascade effect of the generation of new singularities (cf. the experimentation on *bad* NFAs illustrated in Section 6.6).

A more effective notion roughly suggesting the amount of processing required by QUICK SUBSET CONSTRUCTION in comparison with the processing required by SUBSET CONSTRUCTION is the *impact*.

[§§] The software framework is open source and available at https://github.com/MicheleDusi/QuickSubsetConstruction.

[¶¶] Stratified NFAs allow for tuning the degree of nondeterminism. Specifically, varying the *determinism* parameter in different problem configurations is key to varying the amount of singularities processed by QUICK SUBSET CONSTRUCTION, which is directly related to the processing time spent to complete the task. This configuration parameter, however, should not be confused with the notion of *quantity of nondeterminism* introduced in Section 4.1 (cf. Definition 1) which is applicable to any NFA.

Definition 5 (impact). Let $\mathcal{N}$ be an NFA and $\mathcal{D}$ the corresponding SC-equivalent DFA. The *impact* $\mathfrak{I}$ of QUICK SUBSET CONSTRUCTION to transform $\mathcal{N}$ into $\mathcal{D}$ is the ratio between the number n_σ of singularities processed and the number n_δ of transitions in $\mathcal{D}$, namely

$$\mathfrak{I} = \frac{n_\sigma}{n_\delta}. \tag{7}$$

Roughly, the smaller the impact, the more convenient QUICK SUBSET CONSTRUCTION over SUBSET CONSTRUCTION.[##] We can reformulate the statement above as follows: *limited quantity of nondeterminism in the NFA does not necessarily translate into limited impact*.

A precise measure of the benefit (if any) in using QUICK SUBSET CONSTRUCTION is provided by comparing the actual processing time spent by both algorithms in the determinization of the same NFA. The notion of *gain* serves this purpose.

Definition 6 (*gain*). Let t_{SC} and t_{QSC} denote the processing time of SUBSET CONSTRUCTION and QUICK SUBSET CONSTRUCTION, respectively, for generating the DFA that is SC-equivalent to the same NFA. The *gain* $\mathcal{G}$ of QUICK SUBSET CONSTRUCTION, $\mathcal{G} \in [-1 \,..\, 1]$, is the relative portion of processing time that is either saved (if positive) or wasted (if negative) by QUICK SUBSET CONSTRUCTION over SUBSET CONSTRUCTION, namely

$$\mathcal{G} = \frac{t_{SC} - t_{QSC}}{\max\left(t_{SC}, t_{QSC}\right)}. \tag{8}$$

The gain provides a precise information on how much an algorithm outperforms the other. The convenience in using QUICK SUBSET CONSTRUCTION versus SUBSET CONSTRUCTION occurs when the gain is positive: the larger the gain, the better the convenience. Like the impact, the gain too is measured a posteriori, after the termination of both algorithms.[|||]

In the experimentation, five classes of NFAs have been considered, namely *random* NFAs, *acyclic* NFAs, *stratified* NFAs, *weak* NFAs, and *Bad* NFAs. Experimental results for each NFA class considered are shown hereafter.

6.2 | Random NFAs

As the name suggests, a *random* NFA is generated (pseudo-)randomly based on given configuration parameters. We present hereafter the results of three experiments on random NFAs, named R_1, R_2, and R_3, respectively, where each experiment is designed to compare the performance of QUICK SUBSET CONSTRUCTION with that of SUBSET CONSTRUCTION by varying one of two relevant configuration parameters, specifically the number of NFA states and the ε-density, while keeping the other parameters fixed.[***]

The results of each experiment are spread over two graphs (e.g., Figure 12). In the left-hand graph, the x-axis indicates the range of the varying parameter, the y-axis on the left indicates the range of the processing time, and the y-axis on the right indicates the range of the gain (cf. Definition 6 in Section 6.1). Three curves are plotted on that graph: the processing time of SUBSET CONSTRUCTION, the processing time of QUICK SUBSET CONSTRUCTION, and the gain. The processing time indicated corresponds to an average of 100 different executions of the corresponding determinization algorithm (this is true for all the experiments, not only for random NFAs), each one relevant to a different NFA that has been generated based on the same parameter values.

In the second (right-hand) graph, which is specific to QUICK SUBSET CONSTRUCTION only, the x-axis indicates the range of the varying parameter (the same as in the left-hand graph), while the y-axis indicates the range of the singularities processed. Displayed on the graph are the bars indicating the (average) number of singularities processed for each different value of the varying parameter. Each bar incorporates a triangle representing the number of initial singularities. A triangle

[##] The impact, however, can be measured only a posteriori, when QUICK SUBSET CONSTRUCTION terminates; therefore, it cannot be exploited a priori for measuring the convenience in using that algorithm instead of SUBSET CONSTRUCTION, nor does it provide a precise measure of that convenience, as processing a singularity requires in general more time than creating a transition by SUBSET CONSTRUCTION (cf. the discussion in Section 6.8).

[|||] In contrast with the gain, however, the impact requires only the execution of QUICK SUBSET CONSTRUCTION, which provides both the number of singularities processed and the number of transitions in the output DFA.

[***] These parameters are the branching factor (3), the alphabet cardinality (10), and the final density (0.1). Note how increasing the branching factor is bound to increase the quantity of nondeterminism in the NFA, as well as the impact. Conversely, extending the cardinality of the alphabet is likely to decrease the quantity of nondeterminism, as well as the impact. Instead, the percentage of final states is irrelevant to the impact.

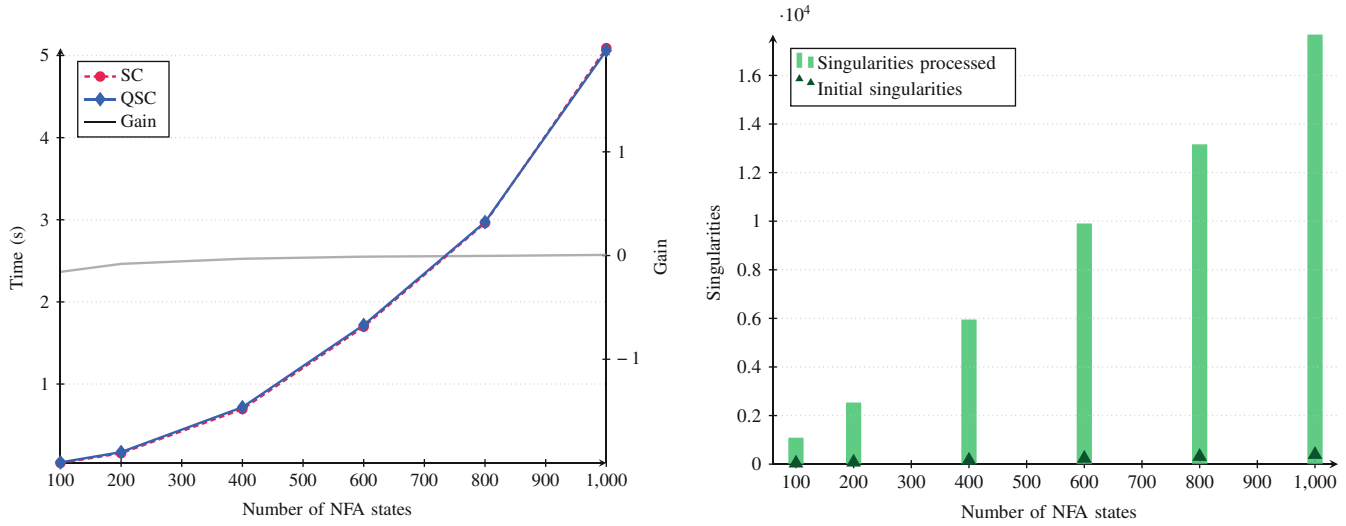

FIGURE 10 Results of experiment R_1 (random NFAs): ε-density = 0 and number of NFA states varying in range [100 .. 1000].

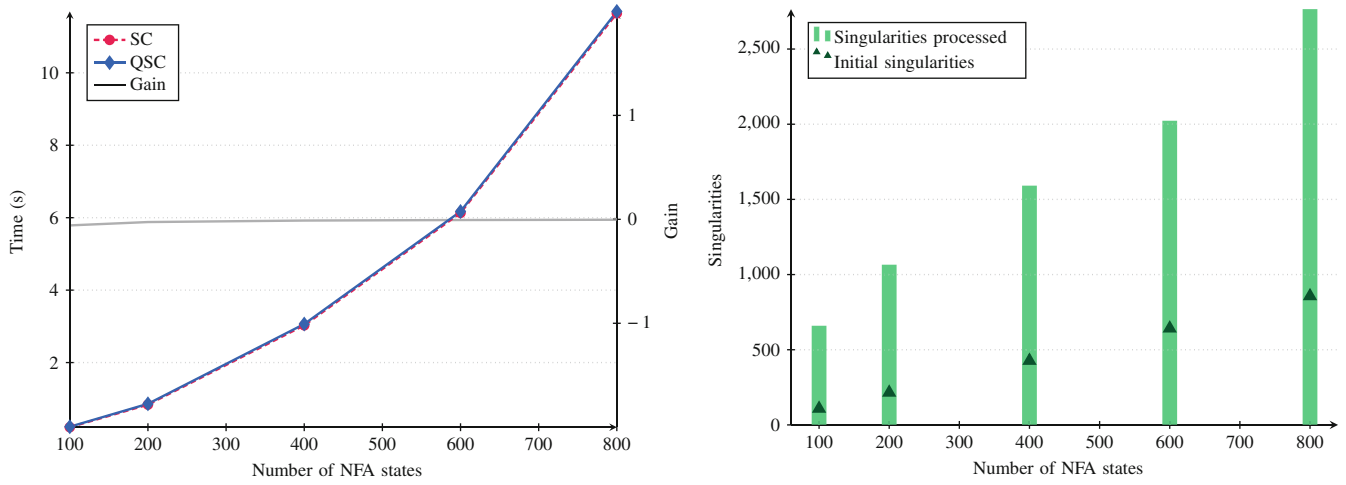

FIGURE 11 Results of experiment R_2 (random NFAs): ε-density = 0.5 and number of NFA states varying in range [100 .. 800].

pointing upwards (as for all bars in Figure 10) indicates that the total number of singularities processed is larger than the number of initial singularities. If, instead, the triangle points downwards, then the actual number of singularities processed is smaller than the number of initial singularities (cf. Figure 17), a condition that may appear inconsistent. The explanation is simple: a singularity (q, ℓ) may be removed from the singularity list before being processed owing to the removal of state q, which comes with the removal of the associated singularities also.

The results of the first experiment (R_1) are displayed in Figure 10, where the varying parameter is the number of NFA states, while no ε-transitions are involved (ε-density = 0). Based on the left-hand graph, QUICK SUBSET CONSTRUCTION compares with SUBSET CONSTRUCTION (the two curves are practically identical, causing the gain to be approximately zero). Based on the right-hand graph in Figure 10, note how the amount of singularities processed by QUICK SUBSET CONSTRUCTION increases linearly with the number of NFA states, which is consistent with an increasing processing time.

Displayed in Figure 11 are the results of the second experiment (R_2), where the varying parameter is again the number of NFA states, while there are 50% of ε-transitions (ε-density = 0.5). Similarly to the first experiment, QUICK SUBSET CONSTRUCTION compares with SUBSET CONSTRUCTION, with the gain being approximately zero. Also, the number of singularities processed grows linearly with the number of NFA states.

Finally, shown in Figure 12 are the results of R_3, the third experiment, where the varying parameter is the ε-density (percentage of ε-transitions), ranging from 0 (no ε-transitions) and 1 (100% of ε-transitions), while the number of NFA

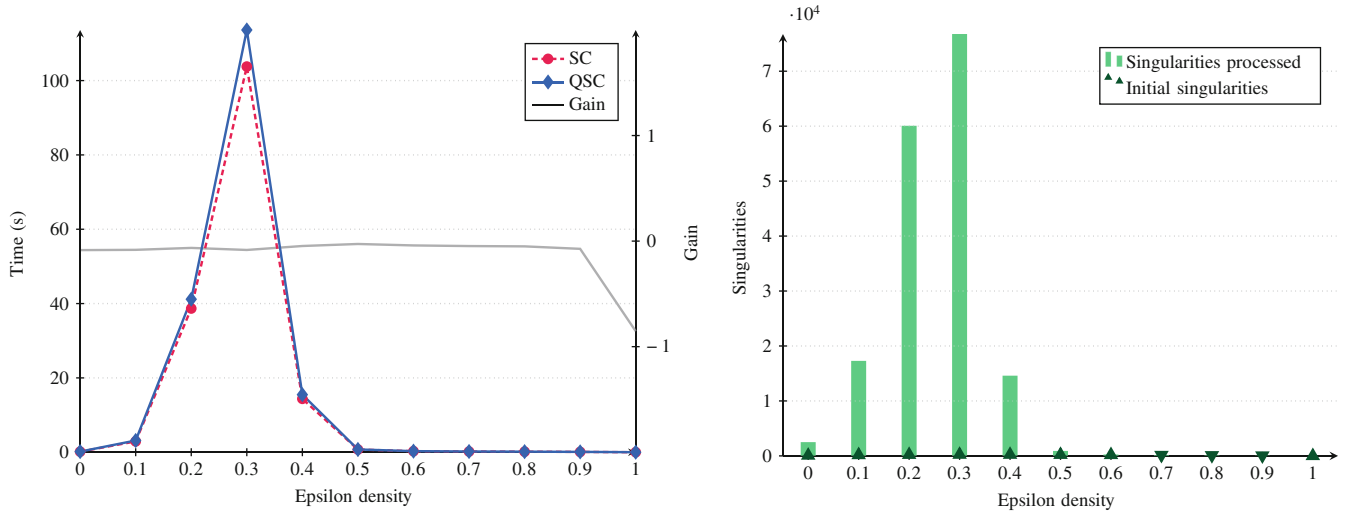

FIGURE 12 Results of experiment R_3 (random NFAs): Number of NFA states = 200 and ε-density varying in range [0 .. 1].

states is fixed (200). In this case, SUBSET CONSTRUCTION performs slightly better than QUICK SUBSET CONSTRUCTION. Both time curves of the algorithms, however, grow monotonically up to a peak (when the ε-density is approximately 0.3), and then decrease almost symmetrically when the ε-density grows further. Unsurprisingly, the trend of the number of singularities processed mimics the trend of the processing time, with a peak when the ε-density is 0.3.

The experimental results presented in Figure 12 raise a natural question: *why the curve of the processing time of both algorithms is spire-shaped when varying the ε-density?* Considering SUBSET CONSTRUCTION, when no ε-transitions exist in the NFA, nondeterminism is caused by multiple transitions with the same label ℓ and exiting the same state. Hence, only the target states of these transitions (the ℓ-mapping) will be embodied in the extension of the corresponding DFA target state. Afterwards, inserting an increasingly percentage of ε-transitions is bound to increase the quantity of nondeterminism and to make the extensions of the DFA states increasingly large, as the ℓ-mapping of a DFA state is likely to include more and more states. This results in a larger number of states as well as a larger number of transitions in the DFA. However, after a certain threshold (about one third of ε-transitions), the extensions of the DFA states are so large that the number of actual subsets keeps decreasing until it becomes just one (when the ε-density is 1), whose extension will include the whole set of NFA states. On the other hand, considering QUICK SUBSET CONSTRUCTION, we need to recall how initial singularities are created when ε-transitions occur. Based on the second rule defined in Section 4.1, if an NFA state n' is exited by an ε-transition, then, for each transition $\langle n, \ell, n' \rangle$ entering n', where $\ell \neq \varepsilon$, a singularity (n, ℓ) is created. Hence, a single ε-transition is bound to cause the creation of several (possibly many) singularities, whose processing make QUICK SUBSET CONSTRUCTION to slow down. Now, consider the bar graph displayed on the right side of Figure 12, indicating the number of singularities processed when varying the ε-density. Unsurprisingly, the number of singularities grows up to the same threshold of ε-density in which the processing time peaks, and then decreases monotonically when the ε-density exceeds that value. The point is, on the one hand, increasing a small ε-density causes the creation of more and more singularities (as pointed out above); on the other hand, however, the larger the number of ε-transitions, the smaller the probability that a state n' exited by an ε-transition is entered by a *non* ε-transition $\langle n, \ell, n' \rangle$, thereby causing a continuous decrease in the number of singularities created and, hence, a shorter and shorter processing time.

In summary, the experimental results presented above suggest that there is no significant difference in the performance of SUBSET CONSTRUCTION and QUICK SUBSET CONSTRUCTION when they are applied to random NFAs. This means that the number of singularities processed is comparable to the number of transitions in the *SC*-equivalent DFA (impact $\simeq 1$).

6.3 | Acyclic NFAs

As the name suggests, acyclic NFAs do not include any cyclic path of transitions. Two experiments have been conducted on acyclic NFAs, namely A_1 and A_2, which are designed to compare QUICK SUBSET CONSTRUCTION with SUBSET CONSTRUCTION by varying either the number of NFA states or the ε-density.

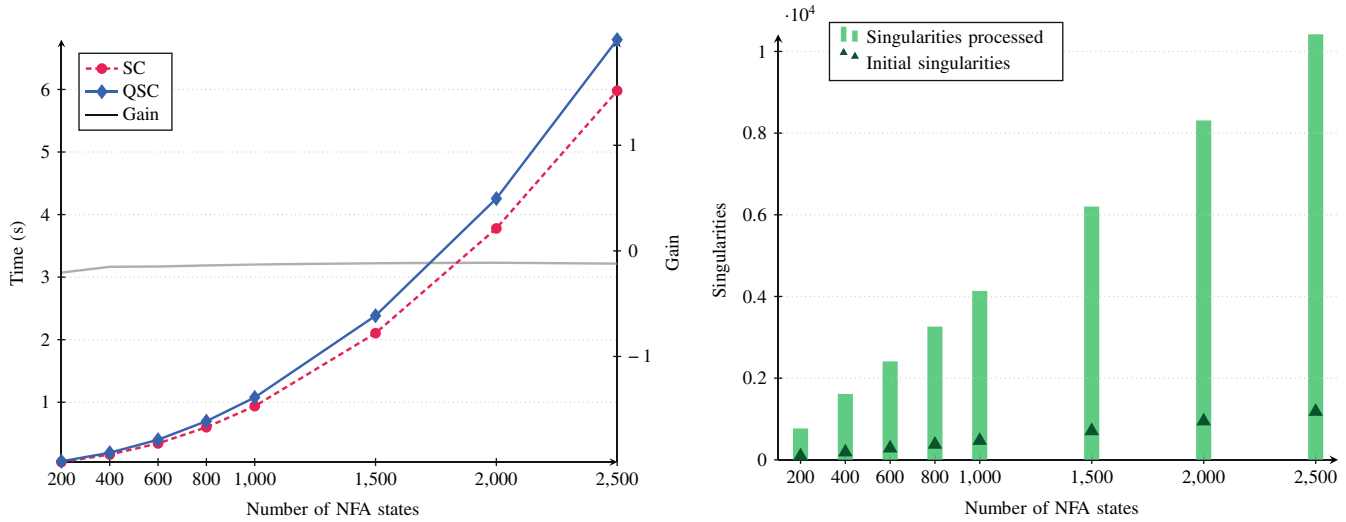

FIGURE 13 Results of experiment A_1 (acyclic NFAs): ε-density = 0 and number of NFA states varying in range [200 .. 2500].

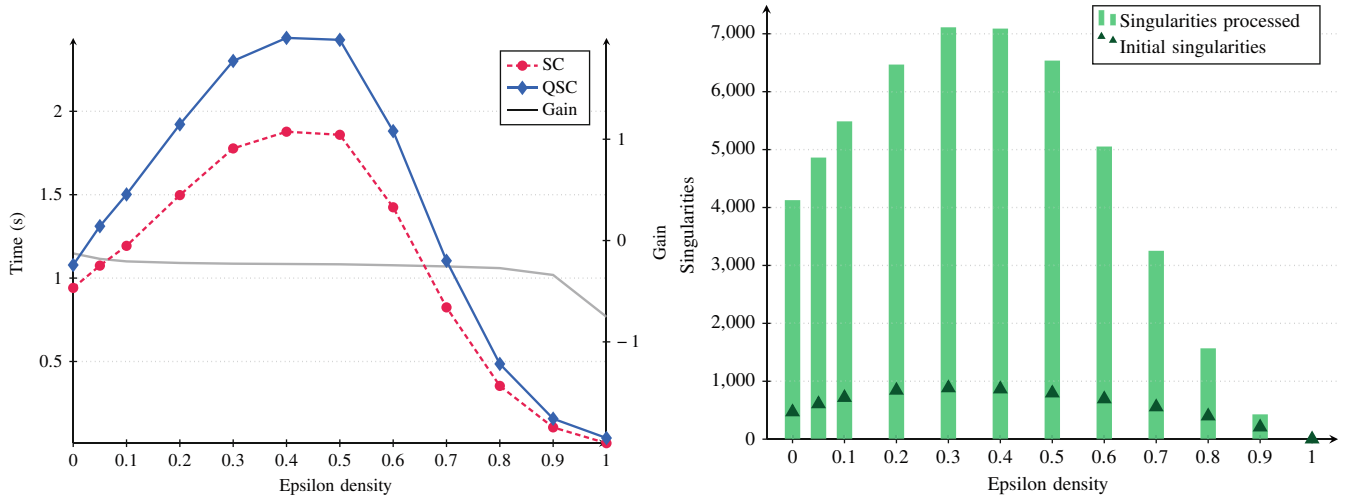

FIGURE 14 Results of experiment A_2 (acyclic NFAs): Number of NFA states = 1000 and ε-density varying in range [0 .. 1].

The results of the fist experiment (A_1) are displayed in Figure 13, where no ε-transitions are involved, while the number of NFA states is varying. Based on the left-hand graph, QUICK SUBSET CONSTRUCTION is invariably outperformed by SUBSET CONSTRUCTION, with an almost constant (negative) gain. As expected, the right-hand graph in Figure 13 shows how the number of singularities processed by QUICK SUBSET CONSTRUCTION follows the trend of its processing time.

Shown in Figure 14 are the results of the second experiment (A_2), where the varying parameter is the ε-density ranging from 0 (no ε-transitions) to 1 (all ε-transitions), while the number of states is kept constant (1000). Even in this experiment, QUICK SUBSET CONSTRUCTION is outperformed by SUBSET CONSTRUCTION, in a way similar to experiment R_3 (random NFAs), with a time peak around the middle of the ε-density. The number of the singularities processed shown in the right-hand graph is consistent with the trend of the processing time of QUICK SUBSET CONSTRUCTION.

The results of the two experiments above suggests that QUICK SUBSET CONSTRUCTION is not convenient when NFAs are acyclic. But why? A plausible explanation is that QUICK SUBSET CONSTRUCTION was conceived for cyclic NFAs. In a sense, this algorithm is oversized when applied to acyclic NFAs, mainly because the determinization of an acyclic NFA does not require the management of the level of states. For example, testing the unsafety of a state (cf. the UNSAFE function in Section 4.3.1) does not require any reasoning on the levels of the parent states: a state is unsafe simply when it is not entered by any other transition. Also, as shown in the left-hand side of Figure 33 in Section 6.8, the impact (cf.

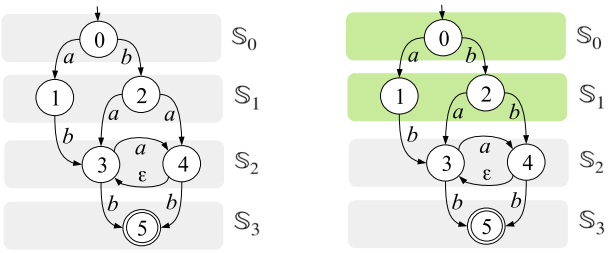

FIGURE 15 Stratified NFAs.

Definition 5) is invariably high (about 0.8), which is bound to make QUICK SUBSET CONSTRUCTION not so convenient regardless of its implementation.

To possibly appreciate QUICK SUBSET CONSTRUCTION over SUBSET CONSTRUCTION, we need to consider NFAs whose determinization is low-impact. Stratified NFAs serve this purpose.

6.4 | Stratified NFAs

Since QUICK SUBSET CONSTRUCTION is bound to outperform SUBSET CONSTRUCTION when the impact is low, that is, when the number of singularities processed is considerably smaller than the number of the transitions in the *SC*-equivalent DFA, we have considered a class of NFAs that allows for an indirect control on the impact, called *stratified* NFAs.

Definition 7 (stratified NFA). Let $\mathcal{N}$ be an NFA, with level of states in the range $[0\,..\,m]$. A *stratum* of $\mathcal{N}$ at level $\overline{\lambda}$, namely $\mathbb{S}_{\overline{\lambda}}$, is the set of states of $\mathcal{N}$ at level $\overline{\lambda}$. The NFA $\mathcal{N}$ is *stratified* if and only if every transition exiting a state s enters a state s' that is either in the same stratum of s or in the next stratum, namely

$$\forall \langle s, \ell, s' \rangle \big(\lambda(s') = \lambda(s) \ \vee \ \lambda(s') = \lambda(s) + 1 \big). \tag{9}$$

The *height* of $\mathcal{N}$ is the number of strata in $\mathcal{N}$, that is $m + 1$.

Example 13 (stratified NFA). Depicted on the left side of Figure 15 is the same NFA shown on the left side of Figure 1. According to Definition 7, this NFA is stratified in four strata, namely $\mathbb{S}_0, \ \ldots \ , \mathbb{S}_3$. Notice how the first stratum $\mathbb{S}_0$ involves just the initial state, a property that holds indistinctly for every stratified NFA.

Proposition 1. *Let $\mathcal{N}$ be a stratified NFA, let $\mathcal{D}$ be the DFA SC-equivalent to $\mathcal{N}$, and let $\langle d, \ell, d' \rangle$ be a transition in $\mathcal{D}$. Then,*

$$\forall n' \in \|d'\| \big(\lambda(n') \geq \min\{ \lambda(n) \mid n \in \|d\| \} \big). \tag{10}$$

This property can be proven by considering that, based on SUBSET CONSTRUCTION, $\|d'\|$ is the ℓ-mapping of $\|d\|$ in $\mathcal{N}$. Owing to the property that a transition exiting a state n in $\mathcal{N}$ cannot reach a state in an upper stratum, all the NFA states in $\|d'\|$ will have a level that cannot be shorter than the minimum level of the states in $\|d\|$.

Based on Proposition 1, it follows that the actual set of states in the *SC*-equivalent DFA of $\mathcal{N}$ is constrained by the condition expressed in (10), which therefore is bound to limit the number of possible states, that is, the size of the equivalent DFA and, ultimately, the impact of QUICK SUBSET CONSTRUCTION. In order to actually control the impact, it suffices confining the nondeterminism to a suffix of the stratified NFA, thereby leaving the rest of the NFA (a prefix of strata) deterministic.

Definition 8 (determinism in a stratified NFA). Let $\mathcal{N}$ be a stratified NFA with height h. The *determinism* of $\mathcal{N}$ is a number Δ, $0 \leq \Delta \leq 1$, defined as follows. If there is a stratum $\mathbb{S}_k$ in $\mathcal{N}$, $0 \leq k < h$, such that the transition function of every state in all strata $\mathbb{S}_0, \ \ldots \ , \mathbb{S}_k$ is deterministic and either $k = h - 1$ or the transition

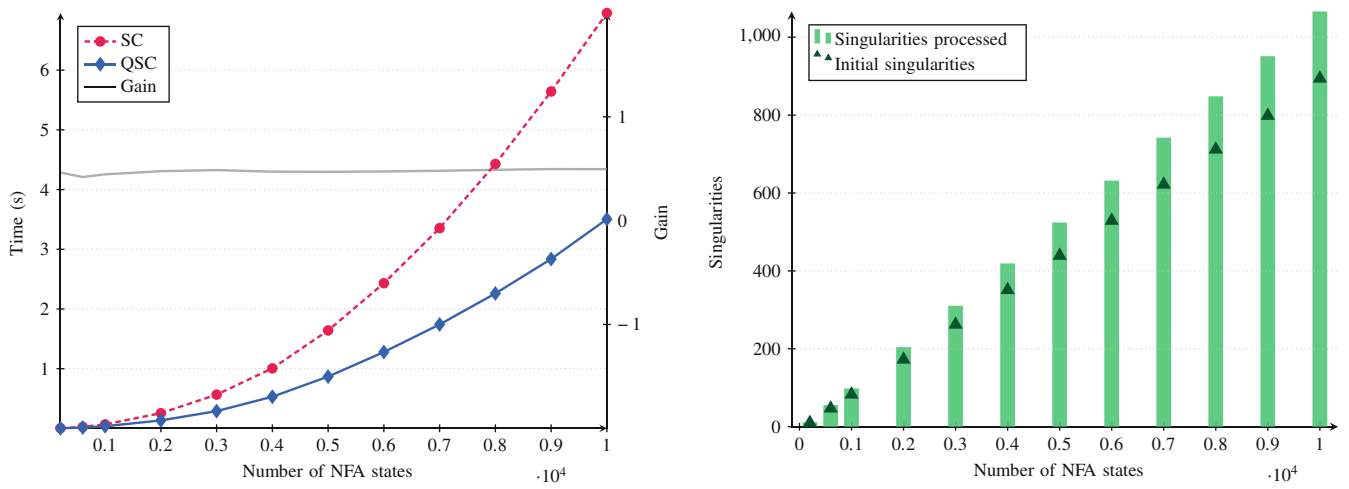

FIGURE 16 Results of experiment S_1 (stratified NFAs): ϵ-density = 0, height = 100, determinism = 0, and number of NFA states varying in range [200 .. 10,000].

function of the states in $\mathbb{S}_{k+1}$ is in general nondeterministic, then

$$\Delta = \frac{k+1}{h}. \tag{11}$$

Otherwise, $\Delta = 0$.

Intuitively, the determinism of $\mathcal{N}$ is the percentage of contiguous strata composing a prefix of $\mathcal{N}$ in which the transition function is deterministic. Consequently, that portion of $\mathcal{N}$ will not involve any singularities, thereby remaining untouched by the execution of QUICK SUBSET CONSTRUCTION.

Example 14 (determinism in a stratified NFA). Shown on the right side of Figure 15 is a slight variation of the NFA on the left side, where the label of the transition $\langle 2, a, 4\rangle$ has been replaced by b. This allows the stratified NFA to have all states in both strata $\mathbb{S}_0$ and $\mathbb{S}_1$ with a deterministic transition function. Based on Definition 8, the determinism of this stratified NFA is

$$\Delta = \frac{k+1}{h} = \frac{1+1}{4} = 0.5. \tag{12}$$

Note how the determinism of the NFA on the left side of Figure 15 is 0.25 as, based on Definition 8, we have $k = 0$.

The degree of determinism in a stratified NFA is key to tuning the impact of QUICK SUBSET CONSTRUCTION. Roughly, based on Proposition 1, the larger the determinism, the smaller the number of states in the *SC*-equivalent DFA and, consequently, the smaller the impact. In other words, we can indirectly have control on the impact by varying the degree of determinism in the stratified NFA.

We present hereafter the results of five experiments on stratified NFAs, named $S_1, \ldots, S_5$, where each experiment is designed to compare the performance of QUICK SUBSET CONSTRUCTION with that of SUBSET CONSTRUCTION by varying one of the relevant configuration parameters, specifically the number of NFA states, the ϵ-density, the height of the NFA (number of strata), and the degree of determinism in the NFA, while keeping the other three parameters fixed.

Displayed in Figure 16 are the results of the first experiment (S_1), where the varying parameter is the number of NFA states, while no ϵ-transitions are involved (ϵ-density = 0). Based on the left-hand graph, QUICK SUBSET CONSTRUCTION outperforms SUBSET CONSTRUCTION with almost constant gain of approximately 0.5. In other words, the execution of QUICK SUBSET CONSTRUCTION takes about 50% less time than SUBSET CONSTRUCTION does to perform the determinization task. Based on the right-hand graph in Figure 16, note how the amount of singularities processed by QUICK SUBSET CONSTRUCTION increases with the number of NFA states.

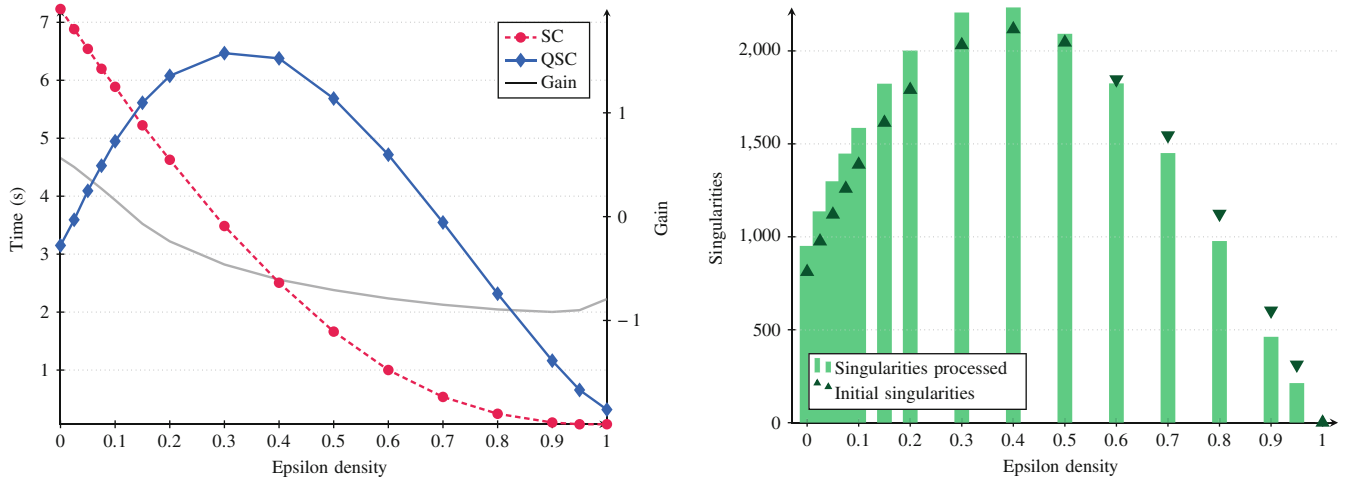

FIGURE 17 Results of experiment S_2 (stratified NFAs): Number of NFA states = 10,000, height = 1000, determinism = 0, and ϵ-density varying in range [0 .. 1].

The results of the second experiment (S_2) are displayed in Figure 17, where the varying parameter is the ϵ-density (percentage of ϵ-transitions), while the number of NFA states is fixed (10,000). In contrast with experiment S_1 (cf. Figure 16), QUICK SUBSET CONSTRUCTION outperforms SUBSET CONSTRUCTION only up to a certain value of ϵ-density (approximately, 15% of ϵ-transitions), beyond which it is increasingly outperformed by SUBSET CONSTRUCTION, as clearly indicated by a negative gain which becomes overwhelmingly so when the ϵ-density is approaching the maximum value (100% of ϵ-transitions).

Note how, in contrast with the results presented in Figure 12 of a similar experiment for random NFAs (R_3), the processing time of SUBSET CONSTRUCTION is very different from that of QUICK SUBSET CONSTRUCTION, as it decreases monotonically with increasing ϵ-density, owing presumably to the particular topology of stratified NFAs. In fact, according to Proposition 1, the extension of the states in the *SC*-equivalent DFA is constrained by condition (9), which limits the number of possible subsets, a property that does *not* hold for random NFAs. Hence, increasing the percentage of ϵ-transitions is bound to generate in the equivalent DFA a smaller number of states (with larger extension), as well as a smaller number of transitions. By contrast, the number of initial singularities created according to the second rule in Section 4.1 and processed by QUICK SUBSET CONSTRUCTION does not depend on the stratification of the NFA and, therefore, a wave-shaped curve (substantially similar to the spire-shaped curve in Figure 12) is produced for the same reasons discussed in Section 6.2 for experiment R_3. Also, based on the bar graph displayed on the right side of Figure 17, note how there are executions of QUICK SUBSET CONSTRUCTION in which the number of singularities processed is smaller than the number of initial singularities, meaning that a subset of the initial singularities are removed before being processed owing to the removal the states associated.

Shown in Figure 18 are the results of S_3, the third experiment, where the varying parameter is the height (ranging in [10 .. 10,000]), while the number of NFA states is fixed (10,000), as well as the ϵ-density (0.5). According to the left-hand graph, QUICK SUBSET CONSTRUCTION is outperformed by SUBSET CONSTRUCTION, with approximately a constant negative gain of about −0.6. Apparently, the large percentage of ϵ-transitions (half of all transitions) is bound to degrade the performance of QUICK SUBSET CONSTRUCTION to such an extent that it is outperformed by SUBSET CONSTRUCTION considerably. This conjecture may be tested by means of a similar experiment in which ϵ-transitions are missing (cf. experiment S_4).

Experiment S_4 is very similar to experiment S_3. The only difference lies in the lack of ϵ-transitions (ϵ-density = 0), as shown in Figure 19. Removing the ϵ-transitions makes QUICK SUBSET CONSTRUCTION to increasingly outperform SUBSET CONSTRUCTION. This is corroborated by the right-hand side diagram, which shows that the number of singularities processed decreases when the height increases, thereby reducing the impact.

The last experiment (S_5) is meant to have control on the impact indirectly by leveraging the degree of determinism in the NFA. As shown in Figure 20, despite a fixed 10% of ϵ-transitions, QUICK SUBSET CONSTRUCTION outperforms

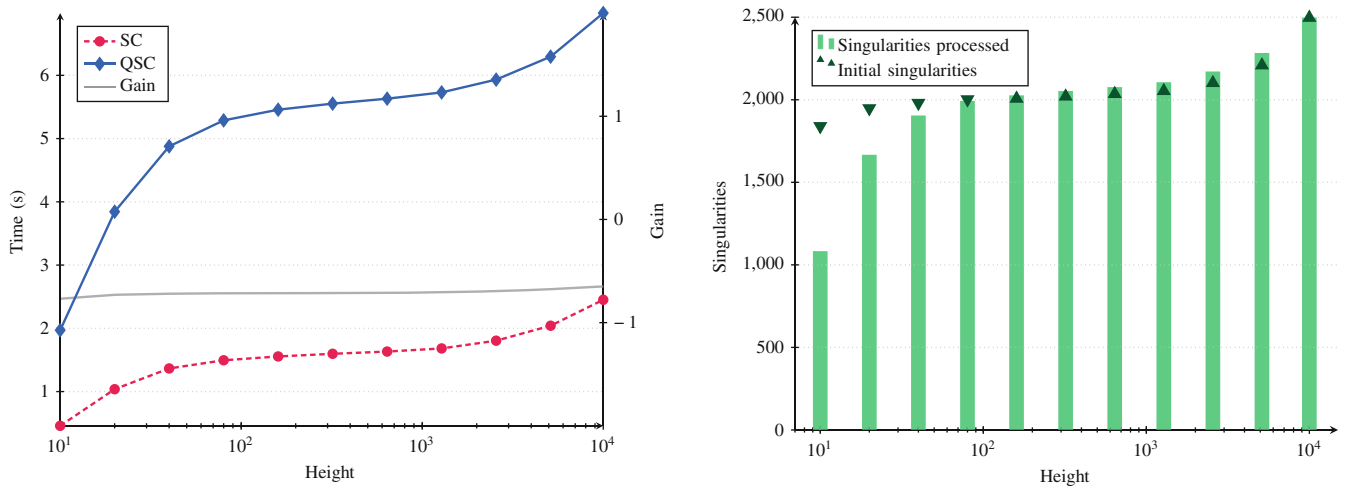

FIGURE 18 Results of experiment S_3 (stratified NFAs): Number of NFA states = 10,000, ϵ-density = 0.5, determinism = 0, and height varying in range [10 .. 10,000].

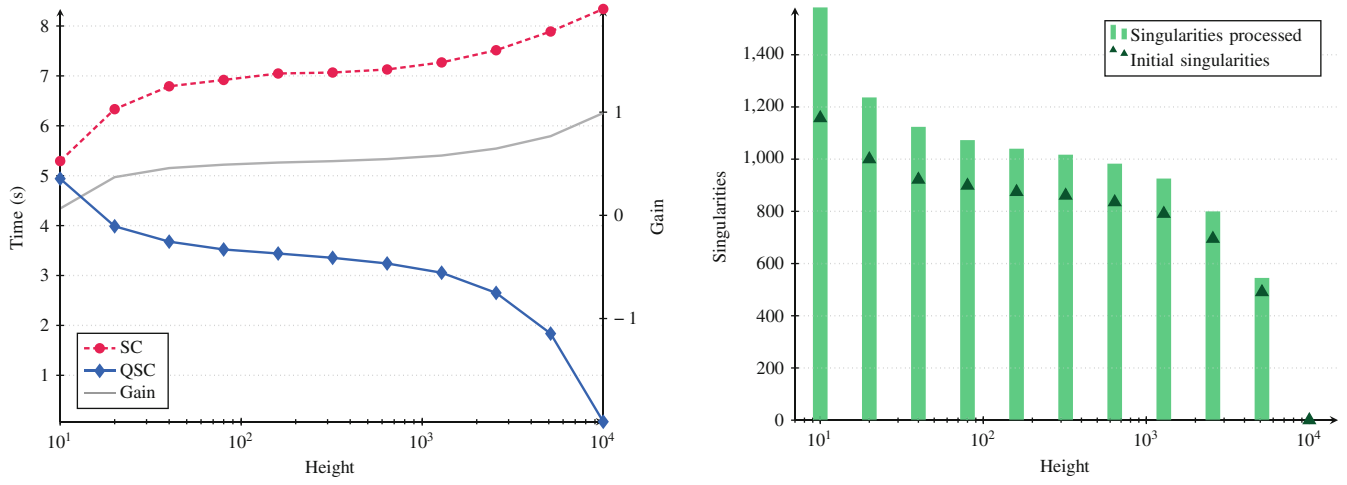

FIGURE 19 Results of experiment S_4 (stratified NFAs): Number of NFA states = 10,000, ϵ-density = 0, determinism = 0, and height varying in range [10 .. 10,000].

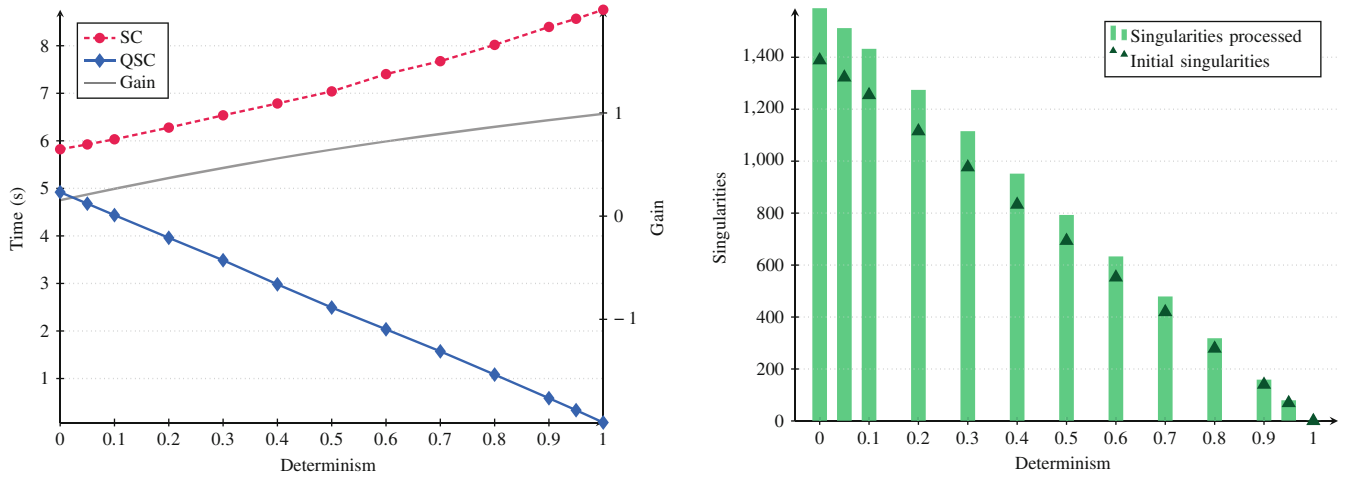

FIGURE 20 Results of experiment S_5 (stratified NFAs): Number of NFA states = 10,000, ϵ-density = 0.1, height = 1000, and determinism varying in range [0 .. 1].

SUBSET CONSTRUCTION with increasing gain. This result is consistent with the right-hand graph, which shows how the number of singularities processed decreases linearly with an increasing determinism. As conjectured above, a stratified NFA with a deterministic prefix of contiguous strata is bound to reduce the set of DFA states because the extension of these states cannot include any NFA state in the deterministic prefix. The larger the determinism, the smaller the impact and, consequently, the shorter the processing time.

In summary, the experimental results for stratified NFAs suggest that QUICK SUBSET CONSTRUCTION outperforms SUBSET CONSTRUCTION when either ε-transitions are missing or determinism in the NFA is high (cf. Definition 8).

In the next section, we consider a different class of NFAs in which the nondeterminism is localized in the transition function of a *single* state, called *weak* NFAs.

6.5 | Weak NFAs

In a *weak* NFA, nondeterminism is either caused by one ε-transition or by two ℓ-transitions exiting the same (single) state. A weak NFA may be generated starting from a DFA that is extended with a single transition exiting a state d with either label ε or with a label ℓ of the alphabet which is the same of another transition already exiting d. In other words, the transition inserted is either $\langle d, \varepsilon, d' \rangle$, where $d' \neq d$, or $\langle d, \ell, d' \rangle$, where $\ell \neq \varepsilon$ and there is another transition $\langle d, \ell, d'' \rangle$ with $d' \neq d''$.

A weak NFA is reminiscent of an environment that is continuously modeled by a learning robot (cf. Section 1), where a DFA (the model of the environment) is extended with some transitions/states that cause the DFA to be transformed into an NFA, thereby requiring on-the-fly determinization of an NFA at each new interaction of the robot with the environment.

The results of three experiments on weak NFAs are presented, namely W_1, W_2, and W_3, where each experiment is designed to compare the performance of QUICK SUBSET CONSTRUCTION with that of SUBSET CONSTRUCTION by varying the number of NFA states, while keeping the other relevant parameters fixed. Since the NFAs are weak, that is, the nondeterminism affects the transition function of a single state, a small impact of QUICK SUBSET CONSTRUCTION is expected and, hence, a large gain.

The results of experiment W_1 are shown in Figure 21, where in each NFA the nondeterminism is caused by a single ε-transition, while the number of NFA states ranges in [200 .. 2500]. While the processing time of SUBSET CONSTRUCTION grows rapidly with the size of the NFA, QUICK SUBSET CONSTRUCTION performs much better, with a processing time growing slowly. This result is corroborated by the limited number of singularities processed (right side of Figure 21), which is of the order of tens, despite the number of states reaching the order of thousands. As a result, the gain is close to 1 (cf. the left side of Figure 21), meaning that QUICK SUBSET CONSTRUCTION is actually much faster than SUBSET CONSTRUCTION.

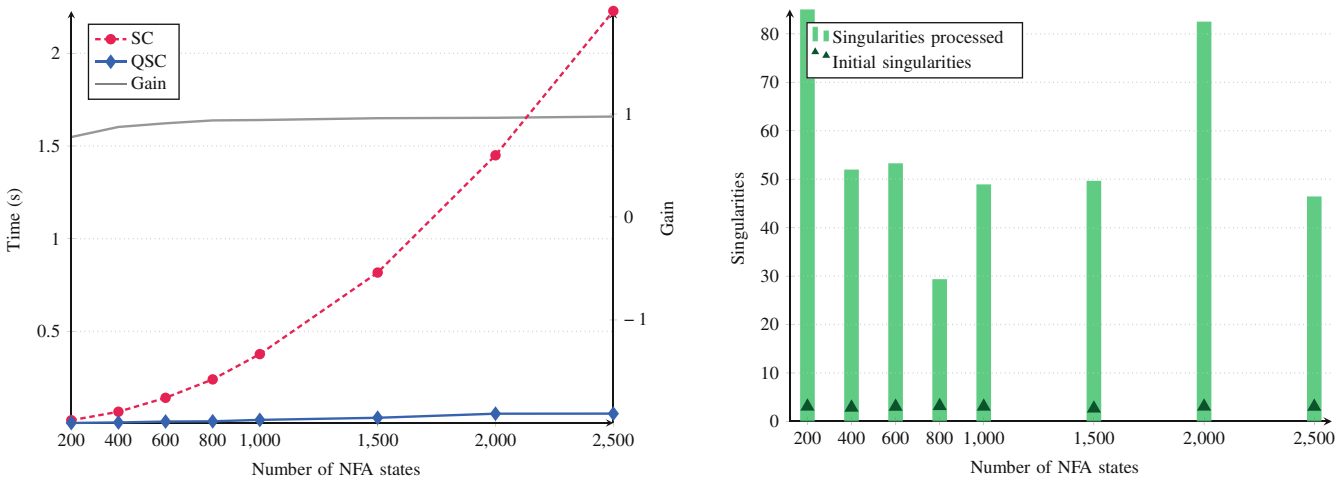

FIGURE 21 Results of experiment W_1 (weak NFAs): One ε-transition, one state with nondeterministic transition function, and number of NFA states varying in range [200 .. 2500].

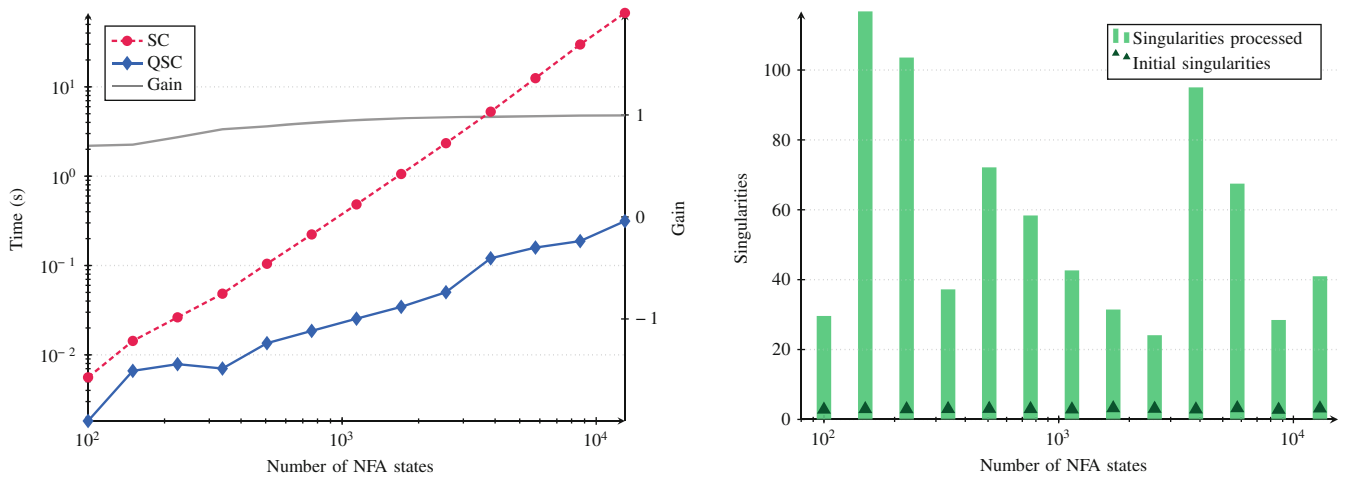

FIGURE 22 Results of experiment W_2 (weak NFAs, processing time in logarithmic scale): One ε-transition, one state with nondeterministic transition function, and number of NFA states varying in range [100 .. 12,975].

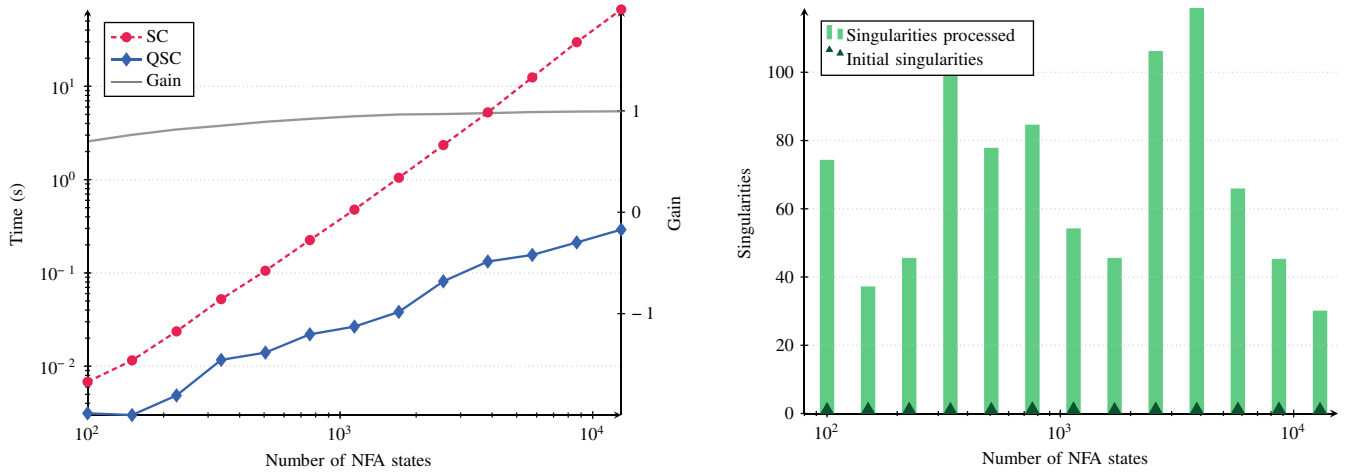

FIGURE 23 Results of experiment W_3 (weak NFAs, processing time in logarithmic scale): ε-density = 0, one state with nondeterministic transition function, and number of NFA states varying in range [100 .. 12,975].

The second experiment (W_2) is similar to W_1, as there is just one single ε-transition causing the nondeterminism in the NFA. Now, however, the range of the number of NFA states is enlarged to [100 .. 12,975]. The results are shown in Figure 22, where the processing time is displayed in *logarithmic* scale. Notice how the time of SUBSET CONSTRUCTION is plotted as a straight line, meaning that it grows exponentially with the size of the NFA. On the other hand, the processing time of QUICK SUBSET CONSTRUCTION too seems to grow approximately linearly, but with a lower slope, thereby resulting in better performance, which is testified by the curve of the gain growing closer and closer to 1. For example, with the largest NFA, the processing time of SUBSET CONSTRUCTION is about 60 s, while the processing time of QUICK SUBSET CONSTRUCTION is about 0.3 s, giving rise to a gain of 0.995, in other words, generating the same equivalent DFA by saving 99.5% of the processing time. As with experiment W_1, the number of singularities processed (right side of Figure 22) is kept within a limited range despite a growing number of NFA states.

The last experiment (W_3) is a variant of experiment W_2, where the ε-transition is substituted by an ℓ-transition, $\ell \neq \varepsilon$, that causes nondeterminism owing to the existence of another ℓ-transition exiting the same state. The results are shown in Figure 23, where the processing time is still in *logarithmic* scale. Note how the evolution of both the processing time and the number of singularities processed is similar to the corresponding evolution in W_2 (cf. Figure 22), indicating that the type of nondeterminism in a weak NFA (ε-transition rather than ℓ-transition) is irrelevant to the performance of the determinization algorithm.

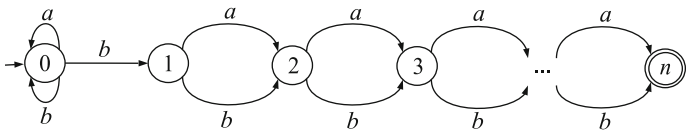

FIGURE 24 Bad NFA, with $n + 1$ states and just one initial singularity $(0, b)$, whose *SC*-equivalent DFA includes 2^n states.

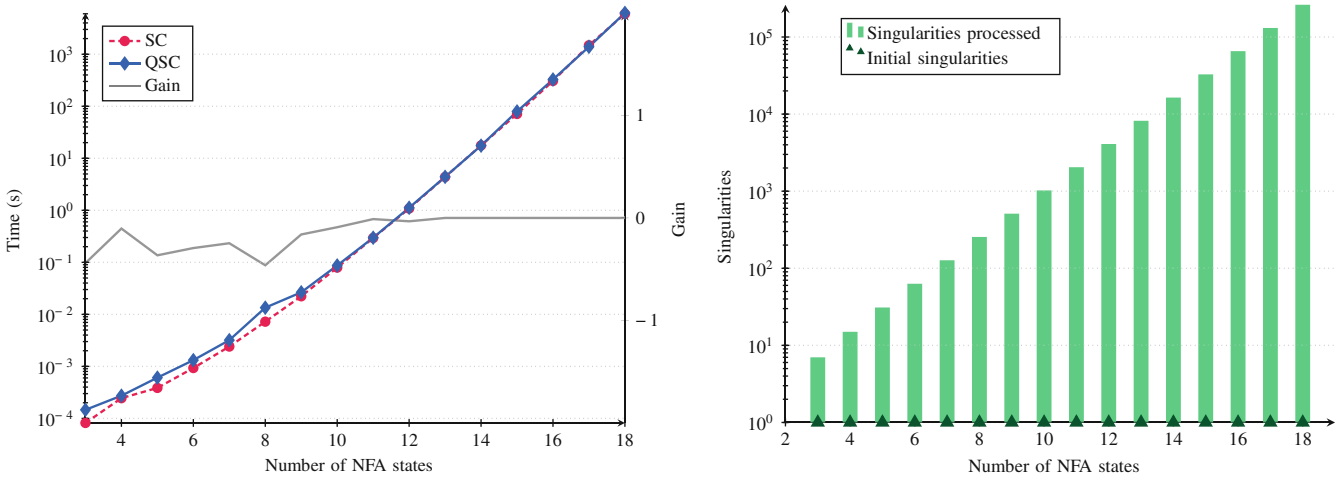

FIGURE 25 Results of the experiment on bad NFAs (processing time and singularities in logarithmic scale): Number of NFA states varying in range [3 .. 18] (hence, number of states in the equivalent DFA ranging in $[2^2 .. 2^{17}] = [4 .. 131{,}072]$).

In summary, these experimental results show that QUICK SUBSET CONSTRUCTION performs much better than SUBSET CONSTRUCTION when applied to weak NFAs.

6.6 | Bad NFAs

When the NFA is large and the impact is small, chances are that QUICK SUBSET CONSTRUCTION outperforms SUBSET CONSTRUCTION considerably. When, instead, the number of singularities processed compares with the number of transitions in the DFA, that is, when the impact is high, QUICK SUBSET CONSTRUCTION is likely to perform similarly to, if not worse than SUBSET CONSTRUCTION. To illustrate this point, we have conducted an experiment on so-called *bad* NFAs. Shown in Figure 24 is one generic such NFA, which includes $n + 1$ states, two alphabet labels, namely a and b, and just one singularity $(0, b)$, irrespective of the actual value of n. Notwithstanding this, it is shown in the literature[†††] that the number of states in the *SC*-equivalent DFA is invariably 2^n, thereby growing exponentially and becoming close to the worst case of $2^{n+1} - 1$ DFA states. Since, by increasing n, the number of DFA states becomes overwhelmingly larger than the number of NFA states, QUICK SUBSET CONSTRUCTION is required to generate the *SC*-equivalent DFA almost entirely as SUBSET CONSTRUCTION does. Consequently, we expect the two algorithms to perform similarly.

The results of the experiment are shown in Figure 25, where the only varying parameter is the number of NFA states, ranging from 3 to 18, in other words, $2 \leq n \leq 17$; hence, the number of states in the *SC*-equivalent DFA will range from 2^2 to 2^{17}. Note how the processing time of the two algorithms (expressed in logarithmic scale) is similar and grows exponentially. This result is corroborated by the bar chart on the right, which clearly show the number of singularities processed growing exponentially (linearly in logarithmic scale).

One may ask why the processing time of QUICK SUBSET CONSTRUCTION is almost identical to that of SUBSET CONSTRUCTION, practically so for $n \geq 10$ (the gain is approximately zero). It looks like the effort in processing a singularity by QUICK SUBSET CONSTRUCTION compares with the effort of generating a transition by SUBSET CONSTRUCTION. Why?

[†††] Compare Reference 9, Section 2.3.6: *A bad case for the subset construction*.

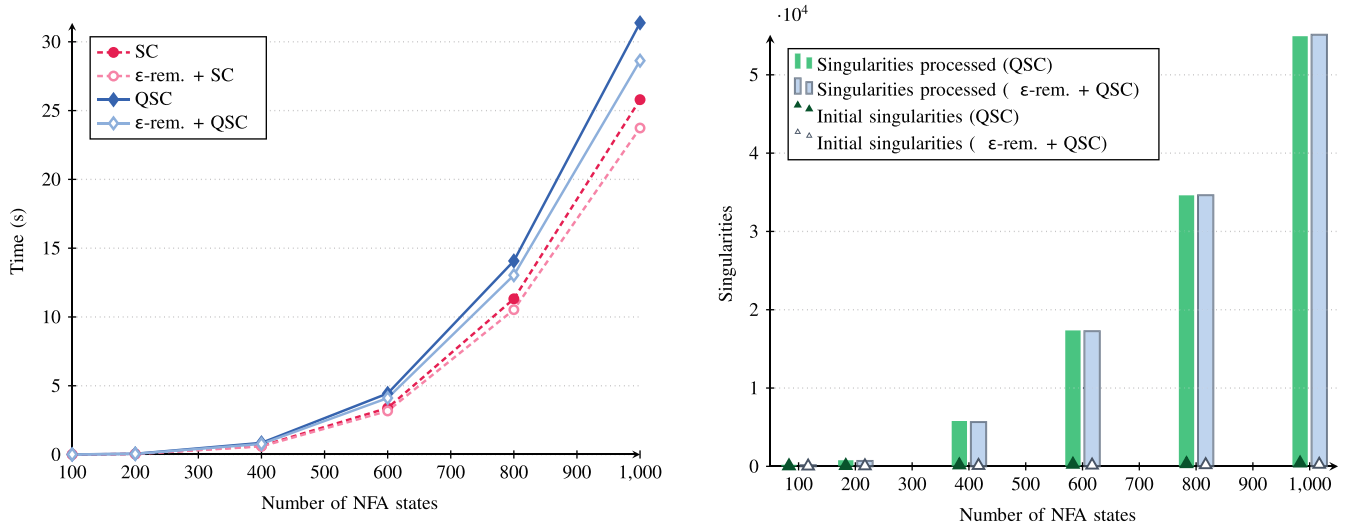

FIGURE 26 Results of experiment P_1 (stratified NFAs with upfront removal of ε-transitions): ε-density = 0.1, height = 20, determinism = 0.5, and number of NFA states varying in range [100 .. 1000].

The reason stems from the actual scenarios involved in the processing of singularities. It turns out that only n singularities are relevant to scenario S_2, while all other singularities (the vast majority of them) are processed in the context of scenario S_1. The point is that processing scenario S_1 roughly mimics the processing by SUBSET CONSTRUCTION, as there is no transition exiting the current state that is marked with the label ℓ involved in the singularity: what is to be done is generating the new transition (marked with ℓ) either directed to an existing state or to a newly-created state, which is exactly what SUBSET CONSTRUCTION does.

6.7 | Upfront removal of ε-transitions

The experimental results presented in Section 6.4 for stratified NFAs clearly indicate that QUICK SUBSET CONSTRUCTION underperforms when the NFA includes ε-transitions: the higher the ε-density, the worse its performance in comparison with SUBSET CONSTRUCTION.[‡‡‡] A natural question arising from this behavior is whether the upfront removal of ε-transitions carried out by a standard polynomial-time algorithm may improve the performance of QUICK SUBSET CONSTRUCTION. To answer this question, five additional experiments have been conducted, named $P_1, \ldots, P_5$, where SUBSET CONSTRUCTION and QUICK SUBSET CONSTRUCTION were run with and without ε-transitions in the NFAs, thereby allowing for a comparison of the processing time in both conditions.

The results of experiment P_1 are shown in Figure 26, where the varying parameter is the number of NFA states. Plotted in the graph on the left side are the curves of the processing time of the two algorithms running on both the original NFA (embodying the ε-transitions) and the NFA resulting from the elimination of the ε-transitions. In the latter case, the processing time represents the sum of the time for the elimination of the ε-transitions (which is independent of the determinization algorithm) and the time for the determinization of the resulting NFA (without ε-transitions). The curves indicate that both algorithms perform slightly better when ε-transitions are removed, with QUICK SUBSET CONSTRUCTION being still outperformed by SUBSET CONSTRUCTION. The bar graph on the right side of the figure shows the corresponding number of singularities processed by QUICK SUBSET CONSTRUCTION in both conditions (with and without ε-transitions), which happen to be substantially the same.

Displayed in Figure 27 are the results of the second experiment (P_2), where the varying parameter is the determinism of the stratified NFA (cf. Definition 8). Note how, with ε-transitions, QUICK SUBSET CONSTRUCTION starts outperforming SUBSET CONSTRUCTION with determinism = 0.2. Without ε-transitions, the performance of SUBSET CONSTRUCTION

[‡‡‡]Very high-densities of ε-transitions do normally not occur in classical application domains (see, for example, the domain of regular expressions considered in Reference 35). By contrast, in our experience, high ε-densities can emerge quite naturally in translated automata.[36]

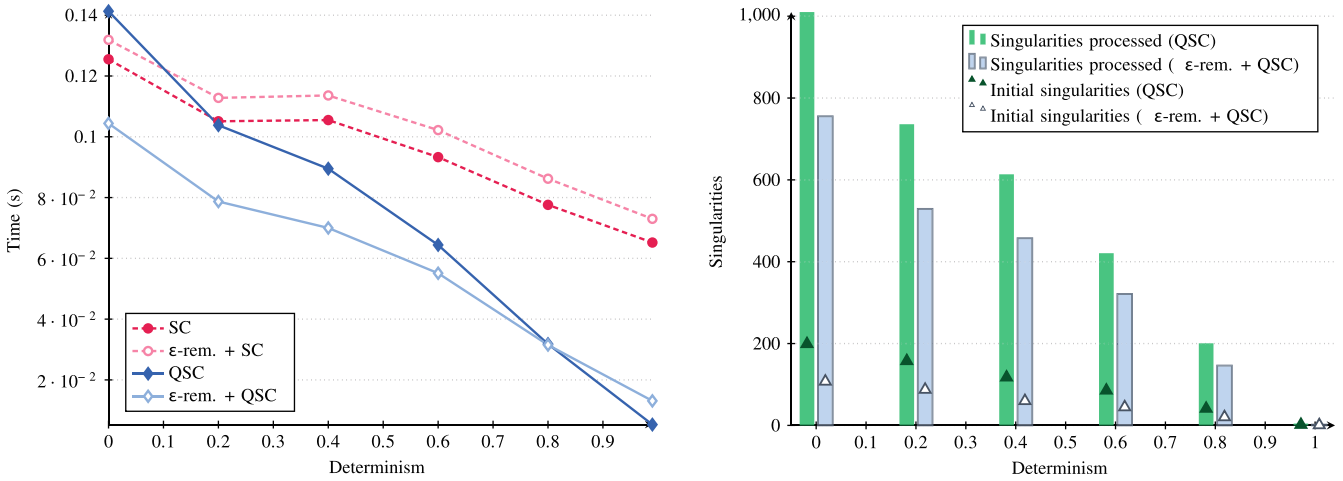

FIGURE 27 Results of experiment P_2 (stratified NFAs with upfront removal of ε-transitions): Number of NFA states = 500, ε-density = 0.1, height = 100, and determinism varying in range [0 .. 1].

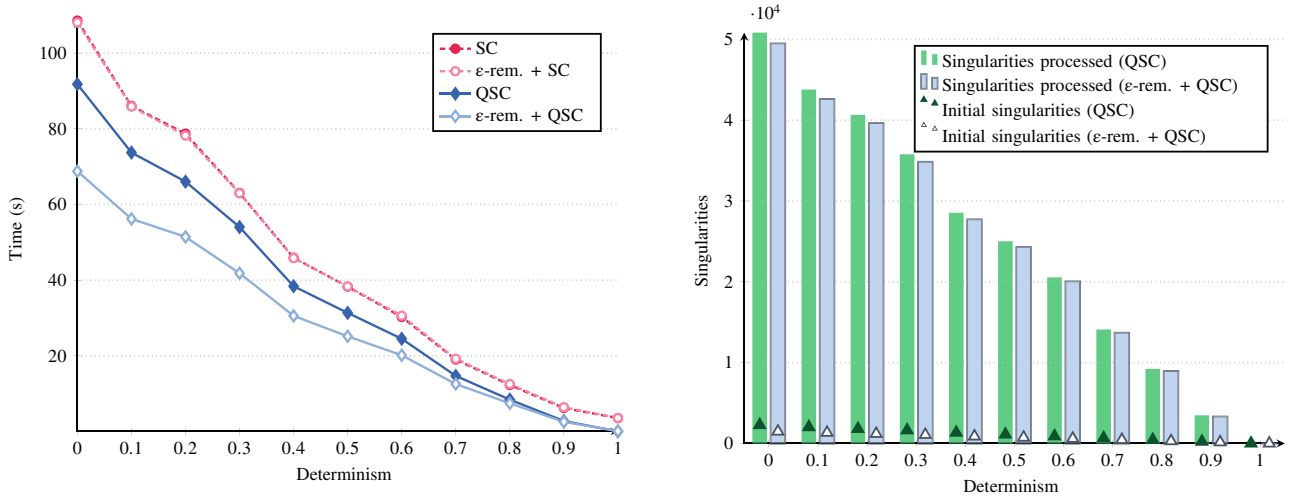

FIGURE 28 Results of experiment P_3 (stratified NFAs with upfront removal of ε-transitions): Number of NFA states = 3000, ε-density = 0.1, height = 1000, and determinism varying in range [0 .. 1].

deteriorates, while that of QUICK SUBSET CONSTRUCTION improves up to determinism = 0.8, remaining however better than the performance of SUBSET CONSTRUCTION for every value of determinism. This improvement is corroborated by the bar graph on the right side of Figure 27, indicating that the number of singularities processed decreases when ε-transitions are removed.

Experiment P_3 is a variation of P_2, with the exception that the NFAs are larger (including 3000 states), with results being shown in Figure 28. Remarkably, the time curves of SUBSET CONSTRUCTION almost coincide in both conditions, while QUICK SUBSET CONSTRUCTION improves its performance when the ε-transitions are removed.

In the next experiment (P_4), the varying parameter is the ε-density. Notice the recurring spire-shaped curve in all four conditions (cf. Figures 12 and 17). These curves indicate that eliminating the ε-transitions causes a considerable mitigation of the bad performance of both algorithms in the subrange of nondeterminism around 0.7, with SUBSET CONSTRUCTION performing slightly better than QUICK SUBSET CONSTRUCTION in almost the entire range.

Even in the last experiment (P_5), the varying parameter is the ε-density, where the NFAs considered are acyclic. As shown in Figure 30, the results are somehow similar to those in experiment P_4 (cf. Figure 29): both algorithms perform better when ε-transitions are removed, at least up to to a threshold of ε-density of about 0.7, with SUBSET CONSTRUCTION still outperforming QUICK SUBSET CONSTRUCTION in all range.

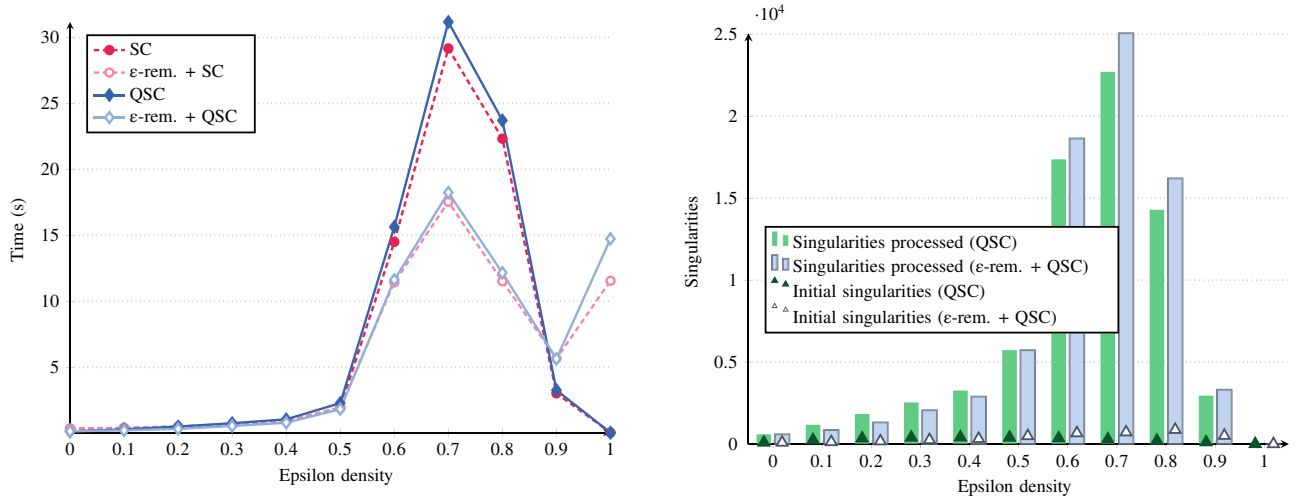

FIGURE 29 Results of experiment P_4 (stratified NFAs with upfront removal of ε-transitions): Number of NFA states = 1000, height = 300, determinism = 0.33, and ε-density varying in range [0 .. 1].

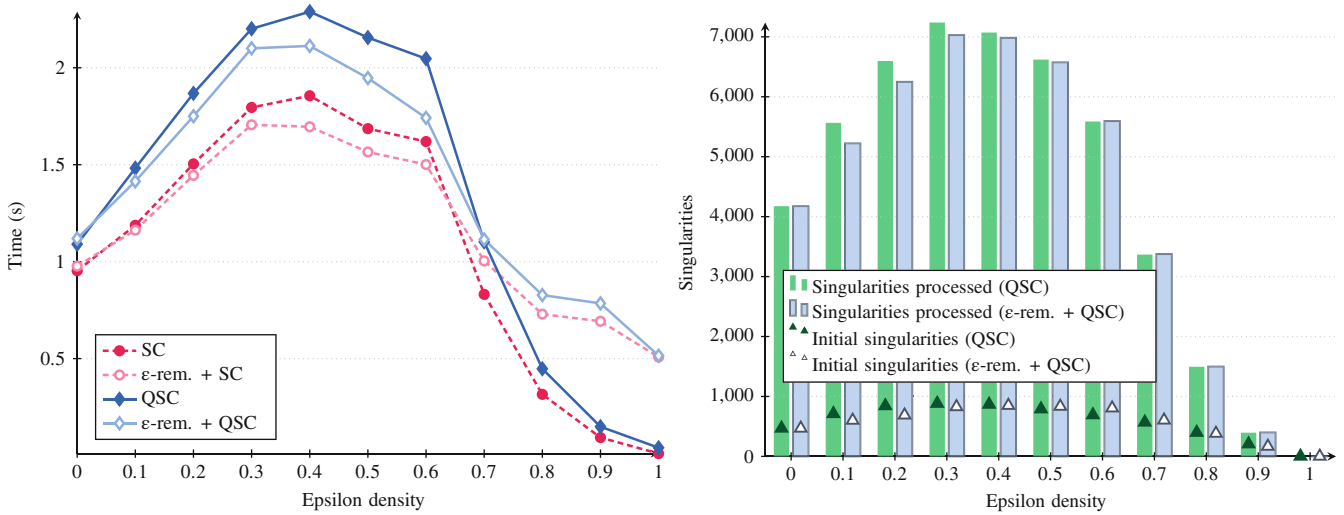

FIGURE 30 Results of experiment P_5 (acyclic NFAs with upfront removal of ε-transitions): Number of NFA states = 1000 and ε-density varying in range [0 .. 1].

In summary, removing the ε-transitions upfront and then applying the determinization process on the resulting NFA seems to improve the performance of both algorithms. Moreover, assuming that the ε-transitions are missing allows for some simplifications of the outward-oriented determinization performed by QUICK SUBSET CONSTRUCTION. First, the first two rules for the creation of the initial singularities are no longer applicable (cf. Section 4.1), thereby leaving just one rule, which applies when a state in the NFA is exited by several ℓ-transitions. Second, scenario S_0 in the algorithm disappears, as the initial state of the NFA cannot be exited by any ε-transitions, thereby leaving only two scenarios: S_1 and S_2. Third, the computation of an ℓ-mapping (e.g., line 21) is simplified as there is no need for the computation of the ε-closure of the set of states reached by the transitions exiting such states with the same label.

6.8 | Is it all about algorithm implementation?

A natural question raised by the experimental results is whether a different implementation of the algorithms might reduce the performance gap between them, possibly in favor of SUBSET CONSTRUCTION. For example, for supporting

the efficiency of QUICK SUBSET CONSTRUCTION, the set of the entering (in addition to the exiting) transitions is maintained for each automaton state.[§§§] Also, the level associated with states is necessary to QUICK SUBSET CONSTRUCTION but completely irrelevant to SUBSET CONSTRUCTION. The processing of this additional data penalize SUBSET CONSTRUCTION, which is forced to compute something both costly and unnecessary. Other "unorthodox" implementation choices might penalize both algorithms, such as identifying a DFA state by a string representing a subset of the NFA states rather than a vector of these states. Intuitively, when both algorithms are equally penalized, however, the performance gap is expected to remain unchanged, although the time response may (equally) deteriorate for both algorithms.[¶¶¶]

In order to test this conjecture in practice, we have developed an improved implementation of SUBSET CONSTRUCTION, namely SUBSET CONSTRUCTION′, where all irrelevant data structures specifically designed for supporting QUICK SUBSET CONSTRUCTION have been removed, thereby resulting in lighter code that has no longer the burden of processing unnecessary information, like the entering transitions or the level attached to each automaton state. Moreover, in order to improve the performance of *both* algorithms, we have also ameliorated the processing of the data structures that are necessary to both SUBSET CONSTRUCTION and QUICK SUBSET CONSTRUCTION, such as the identification of DFA states.

Then, we have performed four experiments, which are reminiscent of experiments R_1 (random NFAs, cf. Figure 10), A_1 (acyclic NFAs, cf. Figure 13), S_1 (stratified NFAs, cf. Figure 16), and W_1 (weak NFAs, cf. Figure 21), where the NFAs have been generated based on the same configuration parameters adopted in the corresponding original experiments.

In each new experiment, SUBSET CONSTRUCTION, SUBSET CONSTRUCTION′, and QUICK SUBSET CONSTRUCTION were run on 100 instances of each NFA (based on the same parameters), eventually plotting the average processing time. The results, shown in Figure 31, clearly indicate that SUBSET CONSTRUCTION is invariably outperformed by SUBSET CONSTRUCTION′, but slightly so. This could make a difference when QUICK SUBSET CONSTRUCTION compares with SUBSET CONSTRUCTION, as in the experiment for random NFAs (A). However, when QUICK SUBSET CONSTRUCTION outperforms SUBSET CONSTRUCTION to a major extent, such as with stratified (C) and weak NFAs (D), code optimization in SUBSET CONSTRUCTION′ is insufficient to make a practical difference.

Comparing the results of each new experiment with those of the corresponding original experiment, notice how despite the substantial similarity of the time curves of SUBSET CONSTRUCTION and QUICK SUBSET CONSTRUCTION, the actual time values are considerably smaller than in the original experiments. This is the effect of code optimization carried out on the data structures that keep being shared by both algorithms.

That said, our claim is that, whatever the implementation technique adopted for either algorithm, the fact remains that QUICK SUBSET CONSTRUCTION may still outperform SUBSET CONSTRUCTION when the impact is below a given threshold; in other words, the better the implementation of SUBSET CONSTRUCTION, the lower the impact is expected to be for QUICK SUBSET CONSTRUCTION to outperform SUBSET CONSTRUCTION, and vice versa.

We first support this claim with the help a so-called *lake-race* metaphor. With reference to Figure 32, imagine a race in which two competing athletes, namely *SC* and *QSC*, are expected to reach a point *finish* starting from a point *start*, both points being on the edge of a lake (blue region). The peculiarity of the race is that there is absolutely no constraint on the chosen path and the moving technique of each athlete. Since it turns out that *SC* is a runner and *QSC* is a swimmer, *SC* is expected to run on the ground (red curved line), while *QSC* prefers swimming in the lake (straight blue line). The question is: *who will win the race?* The answer is not obvious: even if the runner is considerably faster than the swimmer, the path of the swimmer is considerably shorter than the path of the runner; all depends on the actual lengths of the paths and the actual speeds of the athletes.

Let S_{SC} be the length of the path of the runner, let S_{QSC} be the length of the path of the swimmer, let V_{SC} be the (average) speed of the runner, let V_{QSC} be the (average) speed of the swimmer, and let t_{SC} and t_{QSC} be the time of the runner and the swimmer, respectively, to complete the race. The two athletes will reach *finish* at the same time when $t_{SC} = t_{QSC}$, that is, when $S_{SC}/V_{SC} = S_{QSC}/V_{QSC}$, in other words, when $S_{QSC}/S_{SC} = V_{QSC}/V_{SC}$. Hence, denoting with $\mathfrak{I}$ (impact) the ratio S_{QSC}/S_{SC}, whatever the speed of the two athletes, the swimmer will outperform the runner when $\mathfrak{I} < V_{QSC}/V_{SC}$. In other words, the faster the runner, the smaller $\mathfrak{I}$ needs to be in order for the swimmer to outperform the runner and vice versa. For example, assuming $V_{SC} = 5$ m/s and $V_{QSC} = 1.25$ m/s, $\mathfrak{I}$ needs to be less than $1.25/5 = 0.25$; in other words, the

[§§§]Typically, this extra information allows for an efficient check of the condition in line 6 of the UNSAFE auxiliary function (cf. Section 4.3.1).
[¶¶¶]In case of DFA state identification by strings, this is questionable, as comparing two vectors of states is not necessarily more efficient than comparing two strings.

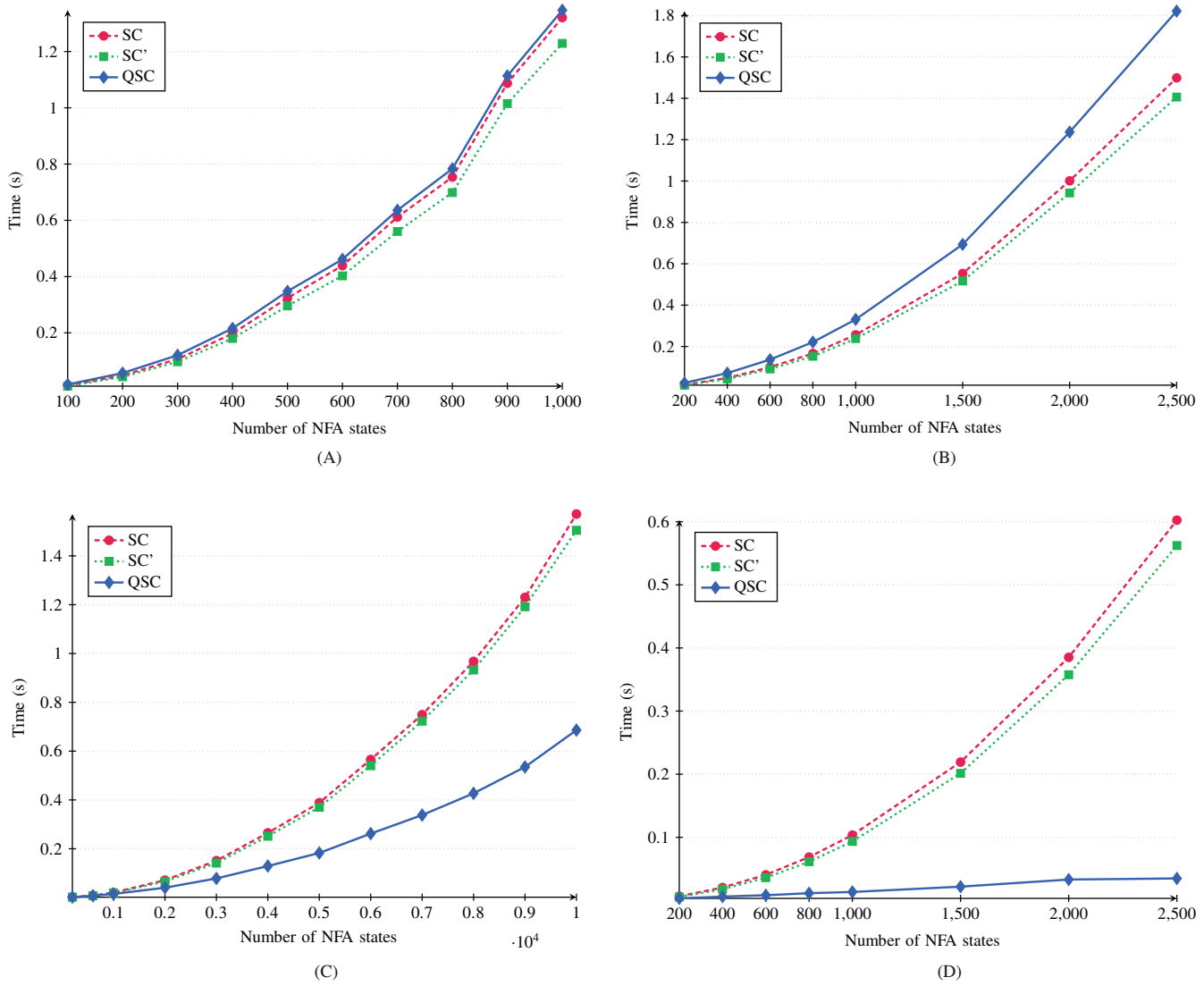

FIGURE 31 Results from extended experimentation with random NFAs (cf. experiment R_1 in Figure 10) (A), acyclic NFAs (cf. experiment A_1 in Figure 13) (B), stratified NFAs (cf. experiment S_1 in Figure 16) (C), and weak NFAs (cf. experiment W_1 in Figure 21) (D), where SC' is a lighter implementation of SUBSET CONSTRUCTION.

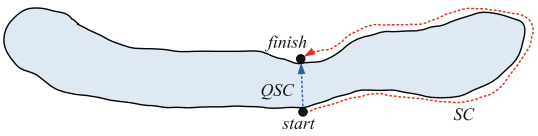

FIGURE 32 The lake-race metaphor: The *QSC* swimmer (blue path on the lake) against the *SC* runner (red path on the ground).

swimmer will outperform the runner provided that the swim path is less than a quarter of the ground path. If the runner increases the speed to 6.25 m/s, then $\mathfrak{T}$ needs to be less than $1.25/6.25 = 0.2$. Subsequently, if the swimmer increases the speed to 1.5 m/s, then $\mathfrak{T}$ needs to be less than $1.5/6.25 = 0.24$.

Out of metaphor, whatever the speed of the two algorithms, which partly depend on their implementation, there will be always a threshold of the impact (cf. Definition 5) under which QUICK SUBSET CONSTRUCTION outperforms SUBSET CONSTRUCTION.

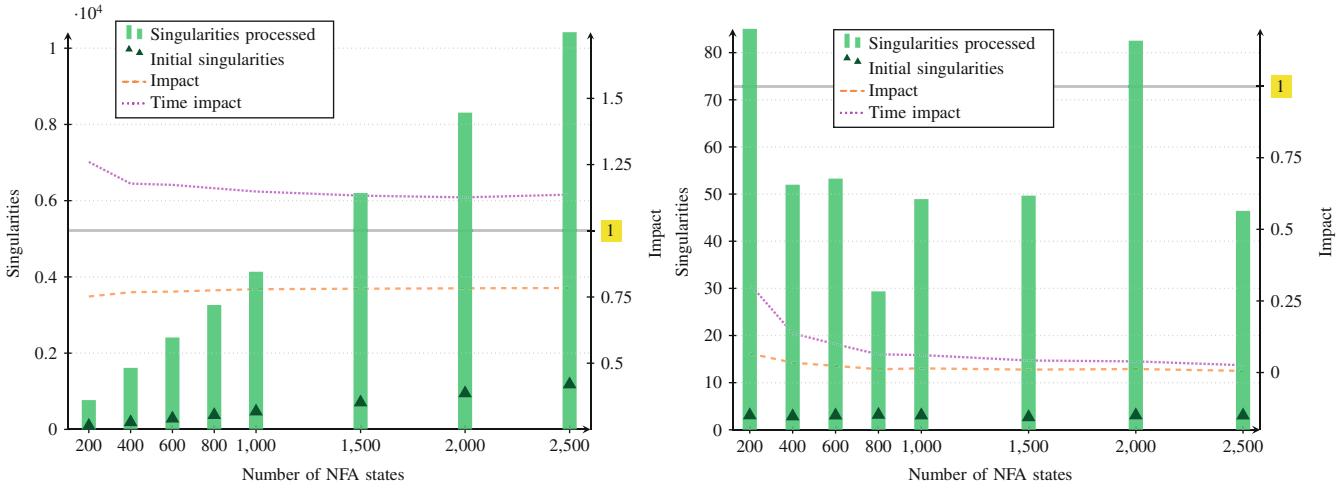

FIGURE 33 Bar charts for experiments A_1 (left) and W_1 (right), which are extended with the impact and the time impact.

Since the impact does not provide a precise measure of the convenience in using QUICK SUBSET CONSTRUCTION, as chances are that processing a singularity requires, on average, more time than creating a transition by SUBSET CONSTRUCTION, we introduce the notion of *time impact*, which provides a precise measure of that convenience by comparing the actual processing time spent by both algorithms in the determinization of the same NFA.

Definition 9 (time impact). Let $\mathcal{N}$ be an NFA and $\mathcal{D}$ the corresponding SC-equivalent DFA. The time impact $\mathfrak{I}_t$ of QUICK SUBSET CONSTRUCTION to transform $\mathcal{N}$ into $\mathcal{D}$ is the ratio between the time t_{QSC} spent by QUICK SUBSET CONSTRUCTION to obtain $\mathcal{D}$ and the time t_{SC} spent by SUBSET CONSTRUCTION to generate $\mathcal{D}$ from scratch, namely

$$\mathfrak{I}_t = \frac{t_{\text{QSC}}}{t_{\text{SC}}}. \tag{13}$$

Clearly, QUICK SUBSET CONSTRUCTION becomes more convenient than SUBSET CONSTRUCTION when $\mathfrak{I}_t < 1$.

Example 15 (impact and time impact). Shown in Figure 33 are the bar charts of the singularities processed in experiments A_1 (left) for acyclic NFAs (cf. Figure 13) and W_1 (right) for weak NFAs (cf. Figure 21), which have been extended with the curves of the impact and the time impact. In the chart on the left, the impact is considerably high (above 0.75). Albeit the impact remains under 1, the time impact is always above 1. Consequently, QUICK SUBSET CONSTRUCTION is invariably outperformed by SUBSET CONSTRUCTION, as already known from Figure 13 (left). By contrast, considering the results in the chart on the right, since the impact is always very low, the time impact remains abundantly under the threshold of 1, thereby indicating that QUICK SUBSET CONSTRUCTION invariably outperforms SUBSET CONSTRUCTION, as already known from Figure 21 (left).

Note how, based on Definition 9, the gain (cf. Definition 6 in Section 6.1) may be expressed in terms of time impact as follows:

$$\mathcal{G} = \begin{cases} \frac{t_{\text{SC}} - t_{\text{QSC}}}{t_{\text{SC}}} = 1 - \frac{t_{\text{QSC}}}{t_{\text{SC}}} = 1 - \mathfrak{I}_t & \text{if } t_{\text{SC}} \geq t_{\text{QSC}} \\ \frac{t_{\text{SC}} - t_{\text{QSC}}}{t_{\text{QSC}}} = \frac{t_{\text{SC}}}{t_{\text{QSC}}} - 1 = \frac{1}{\mathfrak{I}_t} - 1 & \text{otherwise.} \end{cases} \tag{14}$$

Assuming (reasonably) that processing a singularity requires on average more time than generating a transition by SUBSET CONSTRUCTION, we expect $\mathfrak{I}_t > \mathfrak{I}$. The notion of *scale factor* provides a quantitative relation between $\mathfrak{I}_t$ and $\mathfrak{I}$.

Definition 10 (scale factor). Let $\mathfrak{I}$ and $\mathfrak{I}_t$ be the impact and the time impact, respectively, of QUICK SUBSET CONSTRUCTION for the determinization of an NFA. The scale factor is the ratio

$$f = \frac{\mathfrak{I}_t}{\mathfrak{I}}. \tag{15}$$

In other words, based on (7) and (13), the scale factor can be expressed as

$$f = \frac{t_{\mathrm{QSC}} \cdot n_\delta}{t_{\mathrm{SC}} \cdot n_\sigma} = \frac{V_{\mathrm{SC}}}{V_{\mathrm{QSC}}}, \tag{16}$$

where $V_{\mathrm{SC}} = n_\delta / t_{SC}$ is the speed of SUBSET CONSTRUCTION (number of transitions generated per time unit) and $V_{\mathrm{QSC}} = n_\sigma / t_{QSC}$ is the speed of QUICK SUBSET CONSTRUCTION (number of singularities processed per time unit). Intuitively, the scale factor indicates how SUBSET CONSTRUCTION is faster than QUICK SUBSET CONSTRUCTION and, hence, how small the impact needs to be in order for SUBSET CONSTRUCTION to be outperformed by QUICK SUBSET CONSTRUCTION. The notion of *neutral impact* provides a threshold for that convenience.

Definition 11 (neutral impact). The neutral impact is the value $\overline{\mathfrak{I}}$ of the impact that makes the processing time of QUICK SUBSET CONSTRUCTION equal to that of SUBSET CONSTRUCTION in the determinization of an NFA.

Proposition 2. *The neutral impact is the inverse of the scale factor, namely*

$$\overline{\mathfrak{I}} = \frac{1}{f} = \frac{V_{\mathrm{QSC}}}{V_{\mathrm{SC}}}. \tag{17}$$

Proof. Based on Definition 11, the neutral impact is the impact that makes $\mathfrak{I}_t = 1$, namely $\mathfrak{I}_t = \overline{\mathfrak{I}} \cdot f = 1$, that is, $\overline{\mathfrak{I}} = 1/f$. ∎

This means that QUICK SUBSET CONSTRUCTION starts outperforming SUBSET CONSTRUCTION when $\mathfrak{I} < \overline{\mathfrak{I}}$, in other words, when $\mathfrak{I} < V_{\mathrm{QSC}}/V_{\mathrm{SC}}$, a result already known from our lake-race metaphor above.

Example 16 (neutral impact). Assume that SUBSET CONSTRUCTION generates 150 transitions per time unit and QUICK SUBSET CONSTRUCTION processes 90 singularities in the same time unit. The neutral impact is $\overline{\mathfrak{I}} = 90/150 = 0.6$. That is, QUICK SUBSET CONSTRUCTION is more convenient than SUBSET CONSTRUCTION when the number of singularities processed is smaller than 60% of the total number of transitions in the SC-equivalent DFA. If, by code amelioration, we improve the speed of QUICK SUBSET CONSTRUCTION from 90 to 105, the neutral impact will raise to $\overline{\mathfrak{I}} = 105/150 = 0.7$, which allows for increasing the threshold of singularities processed in order to still outperform SUBSET CONSTRUCTION. Subsequently, if we improve the speed of SUBSET CONSTRUCTION from 150 to 175, the neutral impact returns to $\overline{\mathfrak{I}} = 105/175 = 0.6$.

So, *is it all about algorithm implementation?* Apparently not: the faster SUBSET CONSTRUCTION over QUICK SUBSET CONSTRUCTION, the smaller the impact required for QUICK SUBSET CONSTRUCTION to outperform SUBSET CONSTRUCTION. In other words, whatever the quality of the code implementing the two algorithms, in theory, there will be always a threshold of the impact under which QUICK SUBSET CONSTRUCTION remains more convenient than SUBSET CONSTRUCTION.

7 | CONCLUSION

In this article, we presented an algorithm for NFA determinization, named QUICK SUBSET CONSTRUCTION, as an alternative to the classical SUBSET CONSTRUCTION algorithm. Although the two algorithms generate the same (*SC*-equivalent) DFA when applied to the same input NFA (cf. Theorem 1), that DFA is obtained very differently. The approach of SUBSET CONSTRUCTION is outward-oriented: the DFA is generated from scratch, irrespective of the nature of the NFA. The *modus operandi* of QUICK SUBSET CONSTRUCTION, instead, is inward-oriented: the NFA is progressively transformed into the equivalent DFA by applying a series of repair actions on the NFA that remove the nondeterminism by changing the transition function of the states involved.

Repairing an NFA rather than constructing a new DFA entirely may be beneficial in terms of processing time, especially so when the NFA is large and the nondeterminism can be removed by a limited number of repair actions, as the (possibly large) portion of the NFA that is deterministic already may not be subjected to any processing. More precisely, the degree of benefit depends on the impact of QUICK SUBSET CONSTRUCTION (cf. Definition 5), that is, the ratio between the number of singularities processed and the number of transitions in the *SC*-equivalent DFA. Intuitively, the smaller the impact, the faster the processing by QUICK SUBSET CONSTRUCTION. This conjecture has been confirmed by the experimental results presented in Section 6.

Qualifying QUICK SUBSET CONSTRUCTION as *quick* does not mean that this algorithm is invariably faster than SUBSET CONSTRUCTION, as shown by the experimental results. More generally, all depends on the impact, which, on its turn, depends on how the nondeterminism is actually distributed within the NFA. As pointed out, generally speaking, a small quantity of nondeterminism in the NFA does not necessarily translate into a small impact. Hence, QUICK SUBSET CONSTRUCTION needs to be adopted with a grain of salt. Typically, it can be exploited in the application domains in which the impact is supposedly low and just-in-time determinization is required, like we have assumed in the example of the learning robot considered in Section 1.

It should be clear that, in the worst case, the complexity of QUICK SUBSET CONSTRUCTION compares with the complexity of SUBSET CONSTRUCTION, which is exponential in the number of NFA states (see the experiment presented in Section 6.6). This is a consequence of the fact that both algorithms generate the same equivalent DFA. The gain of QUICK SUBSET CONSTRUCTION does not come from less processing on the nondeterministic part of the NFA, which has to be fixed by both algorithms, but rather from the deterministic part of the NFA that remains untouched in the equivalent DFA: the larger this part, the greater the gain.

A practical question that this article does not answer is: given an NFA to be determinized, how can we know *in advance* whether it is more convenient using QUICK SUBSET CONSTRUCTION rather than SUBSET CONSTRUCTION? Put another way: how can we know upfront the impact of QUICK SUBSET CONSTRUCTION? Any heuristics that can answer this question would give the algorithm a more practical strength, as it might suggest the best algorithm in any application context. Given an application domain, two approaches may be envisaged in order to evaluate the convenience in using QUICK SUBSET CONSTRUCTION rather than SUBSET CONSTRUCTION. In the first approach, called *static*, the experimentation of the two algorithms on significant test cases may lead to a Boolean conclusion: either QUICK SUBSET CONSTRUCTION is more convenient than SUBSET CONSTRUCTION or it is not. In the second approach, called *dynamic*, there is no definitive answer, as all depends on the particular NFA at hand. Therefore, the best algorithm needs to be chosen on the fly (dynamically) based on significant characteristics of the NFA, a task that can be supported by techniques based on machine learning.

Another interesting question is whether QUICK SUBSET CONSTRUCTION may be applied to automata with attached data. In principle, if SUBSET CONSTRUCTION is effective in the determinization of these "decorated" automata, then QUICK SUBSET CONSTRUCTION will be equally effective, as the resulting DFAs are identical (cf. Section 5). For instance, in model-based diagnosis of discrete-event systems,[3] a large NFA is generated based on a string of observations of the system, with a diagnosis being attached to each state. When the NFA is determinized, each state d of the equivalent DFA is marked with the set of diagnoses marking the NFA states embodied in d. Hence, both SUBSET CONSTRUCTION and QUICK SUBSET CONSTRUCTION may cope with this sort of determinization. Generally speaking, (QUICK) SUBSET CONSTRUCTION is expected to be effective in the determinization of NFAs with attached data if there is a function that maps the collection of data attached to the NFA states embodied in a DFA state d (resulting from the NFA determinization) to exactly the data required for d. By contrast, when the NFA states are marked with time variables, as in timed automata,[37] determinization is not always possible, as discussed in Reference 38. Even when timed automata are determinizable, a special algorithm is applied, which differs from SUBSET CONSTRUCTION. Consequently, QUICK SUBSET CONSTRUCTION cannot be applied either. However, the fact remains that, whatever alternative outward-oriented determinization algorithm operating on an NFA with attached data, chances are that an equivalent inward-oriented algorithm may be designed along the lines of QUICK SUBSET CONSTRUCTION. This is an interesting topic of future research.

AUTHOR CONTRIBUTIONS

Michele Dusi: Implementation of the open-access software system, experimentation, production of the results, check of the article. **Gianfranco Lamperti:** conceptualization, algorithm design, formal analysis, supervision, writing of the article.

DATA AVAILABILITY STATEMENT

The software system for the experimental results is open source at: https://github.com/MicheleDusi/QuickSubsetConstruction.

ORCID

Gianfranco Lamperti https://orcid.org/0000-0002-1915-6932

REFERENCES

1. Rabin M, Scott D. Finite automata and their decision problems. *IBM J Res Dev*. 1959;3(2):114-125. doi:10.1147/rd.32.0114
2. Hamscher W, Console L, de Kleer J, eds. *Readings in Model-Based Diagnosis*. Morgan Kaufmann; 1992.
3. Lamperti G, Zanella M, Zhao X. *Introduction to Diagnosis of Active Systems*. Springer; 2018.
4. Lamperti G, Scandale M. From diagnosis of active systems to incremental determinization of finite acyclic automata. *AI Commun*. 2013;26(4):373-393. doi:10.3233/AIC-130574
5. Lamperti G, Scandale M, Zanella M. Determinization and minimization of finite acyclic automata by incremental techniques. *Softw Pract Exp*. 2016;46(4):513-549. doi:10.1002/spe.2309
6. Brognoli S, Lamperti G, Scandale M. Incremental determinization of expanding automata. *Comput J*. 2016;59(12):1872-1899. doi:10.1093/comjnl/bxw044
7. Lamperti G. Temporal determinization of mutating finite automata: reconstructing or restructuring. *Softw Pract Exp*. 2020;50:335-367. doi:10.1002/spe.2776
8. Brzozowski J. Canonical regular expressions and minimal state graphs for definite events. *Proceedings of the Symposium on Mathematical Theory of Automata*. MRI Symposia Series. Vol 12. Polytechnic Press, Polytechnic Institute of Brooklyn; 1962:529-561.
9. Hopcroft J, Motwani R, Ullman J. *Introduction to Automata Theory, Languages, and Computation*. 3rd ed. Addison-Wesley; 2006.
10. Friedl J. *Mastering Regular Expressions*. 3rd ed. O'Reilly Media; 2006.
11. Aho A, Lam M, Sethi R, Ullman J. *Compilers—Principles, Techniques, and Tools*. 2nd ed. Addison-Wesley; 2006.
12. Bairoch A, Apweiler R. The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic Acids Res*. 2000;28(1):45-48.
13. Cassandras C, Lafortune S. *Introduction to Discrete Event Systems*. 2nd ed. Springer; 2008.
14. Brave H, Heymann M. Control of discrete event systems modeled as hierarchical state machines. *IEEE Trans Automat Contr*. 1993;38(12):1803-1819.
15. Sampath M, Sengupta R, Lafortune S, Sinnamohideen K, Teneketzis D. Failure diagnosis using discrete-event models. *IEEE Trans Control Syst Technol*. 1996;4(2):105-124.
16. Sampath M, Lafortune S, Teneketzis D. Active diagnosis of discrete-event systems. *IEEE Trans Automat Contr*. 1998;43(7):908-929.
17. Yoo T, Lafortune S. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Trans Automat Contr*. 2002;47(9):1491-1495.
18. Lamperti G, Zanella M. Diagnosis of discrete-event systems from uncertain temporal observations. *Artif Intell*. 2002;137(1–2):91-163. doi:10.1016/S0004-3702(02)00123-6
19. Benveniste A, Fabre E, Haar S, Jard C. Diagnosis of asynchronous discrete-event systems: a net unfolding approach. *IEEE Trans Automat Contr*. 2003;48:714-727.
20. Pencolé Y, Cordier M. A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artif Intell*. 2005;164(1–2):121-170.
21. Su R, Wonham W. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *IEEE Trans Automat Contr*. 2005;50(12):1923-1935.
22. Thorsley D, Teneketzis D. Diagnosability of stochastic discrete-event systems. *IEEE Trans Automat Contr*. 2005;50:476-492. doi:10.1109/TAC.2005.844722
23. Qiu W, Kumar R. Decentralized failure diagnosis of discrete event systems. *IEEE Trans Syst Man Cybern Part A Syst Hum*. 2006;36(2):384-395.
24. Cerutti S, Lamperti G, Scaroni M, Zanella M, Zanni D. A diagnostic environment for automaton networks. *Softw Pract Exp*. 2007;37(4):365-415. doi:10.1002/spe.773
25. Wonham W, Cai K. *Supervisory Control of Discrete-Event Systems*. Communications and Control Engineering. Springer; 2019.
26. Lamperti G, Zanella M, Zhao X. Diagnosis of deep discrete-event systems. *J Artif Intell Res*. 2020;69:1473-1532. doi:10.1613/jair.1.12171
27. Moore E. Gedanken-experiments on sequential machines. In: Shannon CE, McCarthy J, eds. *Automata Studies*. Vol 34. Princeton University Press; 1956:129-153.
28. Hopcroft J. An n log n algorithm for minimizing states in a finite automaton. In: Kohave Z, ed. *The Theory of Machines and Computations*. Academic Press; 1971:189-196.
29. Revuz D. Minimisation of acyclic deterministic automata in linear time. *Theor Comput Sci*. 1992;92(1):181-189.
30. Watson B. *A Taxonomy of Finite Automata Minimization Algorithms*. Computing Science Note. Vol 93/44. Eindhoven University of Technology; 1995.
31. Watson B. A fast new semi-incremental algorithm for the construction of minimal acyclic DFAs. In: Champarnaud JM, Maurel D, Ziadi D, eds. *Automata Implementation*. Lecture Notes in Computer Science. Vol 1660. Springer; 1999:121-132.

32. Watson B. A new recursive incremental algorithm for building minimal acyclic deterministic finite automata. In: Martin-Vide C, Mitrana V, eds. *Grammars and Automata for String Processing: from Mathematics and Computer Science to Biology, and Back: Essays in Honour of Gheorghe Paun*. Topics in Computer Mathematics. Taylor and Francis; 2003:189-202.
33. Watson B, Daciuk J. An efficient incremental DFA minimization algorithm. *Nat Lang Eng*. 2003;9(1):49-64. doi:10.1017/S1351324903003127
34. Almeida M, Moreira N, Reis R. Incremental DFA minimisation. *RAIRO - Theor Inform Appl*. 2014;48:173-186. doi:10.1051/ita/2013045
35. Broda S, Machiavelo A, Moreira N, Reis R. A Hitchhiker's guide to descriptional complexity through analytic combinatorics. *Theor Comput Sci*. 2014;528:85-100.
36. Dusi M, Lamperti G. Conservative determinization of translated automata by embedded subset construction. In: Czarnowski I, Howlett R, Jain L, eds. *Intelligent Decision Technologies*. Smart Innovation, Systems and Technologies. Vol 193. Springer; 2020:49-61.
37. Alur R, Dill D. A theory of timed automata. *Theor Comput Sci*. 1994;126(2):183-235.
38. Baier C, Bertrand N, Bouyer P, Brihaye T. When are timed automata determinizable? In: Albers S, Marchetti-Spaccamela A, Matias Y, Nikoletseas S, Thomas W, eds. *Automata, Languages and Programming*. Lecture Notes in Computer Science. Vol 5556. Springer; 2009:43-54.

How to cite this article: Dusi M, Lamperti G. Quick Subset Construction. *Softw Pract Exper*. 2023;53(11):2092-2132. doi: 10.1002/spe.3246