Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 126 (2018) 59–68

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Knowledge Based and Intelligent Information and Engineering Systems, KES2018, 3-5 September 2018, Belgrade, Serbia

Online Determinization of Large Mutating Automata

Giovanni Caniato*, Gianfranco Lamperti

Department of Information Engineering, University of Brescia, 25123 Brescia, Italy

Abstract

A mutating finite automaton (MFA) is a nondeterministic finite automaton (NFA) which changes its morphology over discrete time by a sequence of mutations, one mutation at each time instant. A mutation involves the insertion and/or removal of a set of states and/or transitions. This results in a sequence of NFAs, one mutated NFA for each mutation. Some application domains, including model-based diagnosis and monitoring of active systems in artificial intelligence and model-based testing in software engineering, require online determinization of MFAs. Determinizing an MFA online means generating a deterministic finite automaton (DFA) as soon as a mutation occurs, which is equivalent to the mutated NFA. Since the classical *Subset Construction* determinization algorithm may be inadequate for MFAs, a conservative algorithm is proposed, called *Subset Restructuring*, that generates the new DFA by restructuring the previous DFA based on the mutation occurred, instead of building it from scratch. Experimental results indicate the effectiveness of the approach, especially so when large MFAs change in time by small mutations.

Keywords: finite automata, determinization, mutating automata, model-based reasoning, discrete-event systems, intelligent monitoring

1. Introduction

Determinization of nondeterministic finite automata (NFAs) is traditionally performed by the *Subset Construction* algorithm [1,2]. Some application domains, such as diagnosis of active systems in artificial intelligence [3,4,5,6,7] and model-based testing in software engineering [8,9], require determinizing an NFA which changes its topology over discrete time $t_0, t_1, \ldots, t_k$. Such a *mutating finite automaton* (MFA) is represented by a pair $(\mathcal{N}_0, \mathcal{M})$, where $\mathcal{N}_0$ is the initial NFA (at time t_0) and $\mathcal{M} = [\mu_1, \ldots, \mu_k]$ is a sequence of *mutations* of the NFA, where each mutation μ_i, occurring at time t_i, $i \in [1\,..\,k]$, is a set of update actions (insertion and/or removal of states and/or transitions). This results in a sequence of *mutated* NFAs, namely $[\mathcal{N}_1, \ldots, \mathcal{N}_k]$, where $\forall i \in [1\,..\,k]$, $\mathcal{N}_i$ is obtained by updating $\mathcal{N}_{i-1}$ by μ_i. Determinizing an MFA online means generating the sequence of deterministic finite automata (DFAs) as soon as mutations occur, where each DFA is equivalent to the corresponding mutated NFA. After some pioneering approaches [10,11,12], the problem of online determinization of MFAs has been partially solved under the restricted assumptions of *incremental* mutations on the one hand [13,14], and for *decremental* mutations [15] (with no experimental results) on the other. Although a conspicuous set of works for incremental processing of finite automata exists in the literature, including [16,17,18,19,20,21],

* Giovanni Caniato. Tel.: +39-030-371-5491 ; fax: +39-030-380-014.
E-mail address: g.caniato@studenti.unibs.it

10.1016/j.procs.2018.07.209

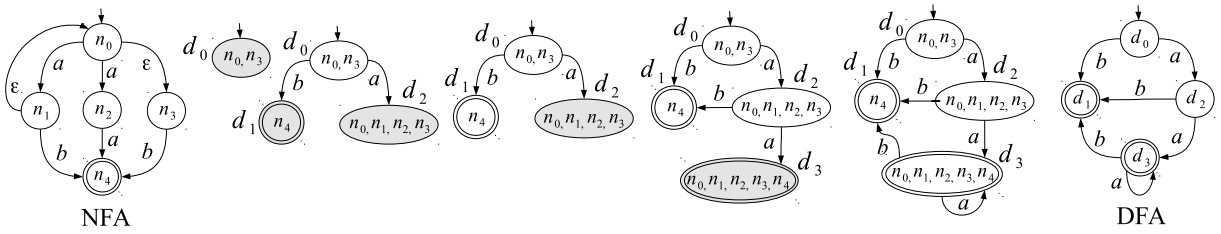

Fig. 1. An NFA (left), an equivalent DFA (right), and the intermediate steps of *Subset Construction* (in between).

nonetheless these works are not designed for (hence, do not solve) the problem of MFA determinization. Since the performance of MFA determinization via *Subset Construction* may deteriorate when NFAs are large and mutations are small, this paper introduces *Subset Restructuring*, an algorithm for determinization of MFAs, where mutations involve both incremental and decremental update actions. The main contributions of this work are: (1) integration of the algorithms proposed in [14,15] into a single new algorithm, namely *Subset Restructuring*, (2) implementation of *Subset Restructuring*, and (3) experimentation aimed at showing the correctness and effectiveness of the algorithm.

2. Finite automata

A finite automaton (FA) can be either deterministic (DFA) or nondeterministic (NFA). A DFA is a 5-tuple $(D, \Sigma, T_\mathrm{d}, d_0, F_\mathrm{d})$, where D is the set of states, Σ is a finite set of symbols called the alphabet, T_d is the transition function, $T_\mathrm{d} : D_\Sigma \mapsto D$, where $D_\Sigma \subseteq D \times \Sigma$, d_0 is the initial state, and $F_\mathrm{d} \subseteq D$ is the set of final states. Determinism comes from T_d mapping a state-symbol pair into a *single* state. An NFA is a 5-tuple $(N, \Sigma, T_\mathrm{n}, n_0, F_\mathrm{n})$, where the fields have the same meaning as in the DFA except that the transition function is nondeterministic, $T_\mathrm{n} : N^\varepsilon_\Sigma \mapsto 2^N$, where $N^\varepsilon_\Sigma \subseteq N \times (\Sigma \cup \{\varepsilon\})$, with ε being the *empty* symbol, $\varepsilon \notin \Sigma$. Each FA is associated with a regular language, which is the set of strings on Σ generated by a path from the initial state to a final state. Two FAs are *equivalent* when they are associated with the same regular language. Within an FA, a transition mapping a pair (s, ℓ) is said to be *marked* by the symbol ℓ and is called an *ℓ-transition*.

Example 1. Displayed on the left side of Fig. 1 is the diagram of an NFA, where n_0 is the initial state, $\Sigma = \{a, b\}$, and $F_\mathrm{d} = \{n_4\}$. The transition function is represented by arcs $n \xrightarrow{\ell} n'$ denoting single transitions, where n' is one of the states mapped from $(n, \ell) \in (N \times (\Sigma \cup \{\varepsilon\}))$. Nondeterminism of the NFA is 3-fold: two ε-transitions and the state n_0 being exited by two a-transitions. The regular language of the NFA includes string ab, which is generated by either path $n_0 \xrightarrow{a} n_1 \xrightarrow{b} n_4$ or $n_0 \xrightarrow{a} n_1 \xrightarrow{\varepsilon} n_0 \xrightarrow{\varepsilon} n_3 \xrightarrow{b} n_4$, as ε is immaterial. The fact that the same string can be generated by several paths comes from the nondeterminism of the NFA. By contrast, in a DFA, each string in the language is generated by just one path. Shown on the right side of Fig. 1 is a DFA equivalent to the NFA, with d_0 being the initial state, and d_1 and d_3 the final states.

Definition 1. *Let $\mathbb{N}$ be a set of states in an NFA and ℓ a symbol in the alphabet. The ε-closure of $\mathbb{N}$ is the union of $\mathbb{N}$ and the set of states that are reached by a path of transitions exiting a state in $\mathbb{N}$, where all such transitions are marked by ε. The ℓ-closure of $\mathbb{N}$ is the ε-closure of the set of states reached by the ℓ-transitions leaving states in $\mathbb{N}$.*

Example 2. In the NFA in Fig. 1, we have ε-closure$(\{n_0, n_1\}) = \{n_0, n_1, n_3\}$ and a-closure$(\{n_0, n_2\}) = \{n_0, n_1, n_2, n_3, n_4\}$.

For each NFA there exists an equivalent DFA that can be generated by *Subset Construction*. Outlined in Algorithm 1 is a pseudo-code specification of *Subset Construction*, where $\mathcal{N}$ is the input NFA and $\mathcal{D}$ is the output DFA. By construction, each state in the DFA is identified by a subset of the states of the NFA (hence the name of the algorithm). The subset of states of the NFA identifying a state d of the DFA is denoted $\|d\|$. The initial state d_0 of the DFA is the ε-closure of the initial state n_0 of the NFA. The DFA is generated with the support of a stack $\mathcal{S}$. Each element in $\mathcal{S}$ is a state of the DFA to be processed (its transition function is to be generated). At the beginning, $\mathcal{S}$ contains the initial state d_0 only. Then, *Subset Construction* pops and processes one state d from $\mathcal{S}$ at a time, by generating the transitions

Algorithm 1 *Subset Construction*

procedure SUBSET CONSTRUCTION($\mathcal{N}$, $\mathcal{D}$)
  $\mathcal{N} = (N, \Sigma, T_n, n_0, F_n)$: an NFA
  $\mathcal{D} = (D, \Sigma, T_d, d_0, F_d)$: the DFA equivalent to $\mathcal{N}$

  Generate the initial state d_0 where $||d_0|| = \varepsilon$-closure(n_0)
  $\mathcal{S} \leftarrow [d_0]$
  repeat
    Pop a state d from $\mathcal{B}$
    for all $\ell \in \Sigma$ such that $n \xrightarrow{\ell} n' \in T_n, n \in ||d||$ **do**
      $\mathbb{N} := \ell$-closure($||d||$)
      if there is no state d' in D such that $||d'|| = \mathbb{N}$ **then**
        Create a new state d' where $||d'|| = \mathbb{N}$
        Push d' onto $\mathcal{S}$
      else
        Let d' be the state in $\mathcal{D}$ such that $||d'|| = \mathbb{N}$
      end if
      Create a new transition $d \xrightarrow{\ell} d'$
    end for
  until $\mathcal{S}$ is empty
end procedure

exiting d, until $\mathcal{S}$ becomes empty. Each transition is generated by considering each label $\ell \in \Sigma$ marking a transition exiting a state $n \in ||d||$. For each label ℓ, a transition $d \xrightarrow{\ell} d'$ is created, where $||d'|| = \ell$-closure($||d||$). Furthermore, if d' does not exist, then it is created (and possibly qualified as final, if it contains at least one state that is final in the NFA), and pushed onto stack $\mathcal{S}$.

Determinization is convenient because processing a DFA is generally more efficient than processing an NFA. For instance, in lexical analysis, the recognition of a string based on a DFA is performed without backtracking[22]. The DFA generated by *Subset Construction* by determinization of an NFA $\mathcal{N}$ is said to be SC-*equivalent* to $\mathcal{N}$.

Example 3. Consider the NFA displayed on the left-hand side of Fig. 1. Traced next to the NFA are the intermediate representations of the equivalent DFA generated by *Subset Construction*, where current states in stack $\mathcal{S}$ are shadowed.

3. Mutating determinization problems

Before presenting *Subset Restructuring*, we give the notions of a mutation and a mutating determinization problem.

Definition 2. *Let $\mathcal{N} = (N, \Sigma, T, n_0, F)$ be an NFA. A* mutation μ *of $\mathcal{N}$ is a triple $(\Delta N, \Delta T, \Delta F)$, where ΔN is the set of mutated states, ΔT the set of mutated transitions, and ΔF the set of mutated final states. Each mutated element is prefixed by either sign "+" or "−", meaning that it is either added to or removed from $\mathcal{N}$, respectively. The* mutated *NFA resulting from the application of μ to $\mathcal{N}$ is denoted $\mu(\mathcal{N})$.*

The following constraints are assumed: (*a*) $\mu(\mathcal{N})$ has the same initial state n_0 as $\mathcal{N}$; (*b*) $\{-n, +n\} \not\subseteq \Delta N$ (a state cannot be both removed and inserted); (*c*) if $-n \in \Delta N, n \in F$, then $-n \in \Delta F$; (*d*) both $\mathcal{N}$ and $\mu(\mathcal{N})$ include at least one final state; (*e*) in both $\mathcal{N}$ and $\mu(\mathcal{N})$ each state is reachable from n_0 and can reach at least one final state.

Definition 3. *Given a DFA $\mathcal{D}$* SC*-equivalent to an NFA $\mathcal{N}$ and a mutation μ of $\mathcal{N}$, the* mutating determinization problem *based on $\mathcal{N}$, $\mathcal{D}$, and μ consists in generating the DFA $\mathcal{D}'$* SC*-equivalent to the mutated NFA $\mu(\mathcal{N})$.*

Example 4. Displayed on the left side of Fig. 2 is a mutated NFA $\mu(\mathcal{N})$, where $\mathcal{N}$ is represented by plain and dashed lines, while μ corresponds to dashed lines (removals) and bold lines (insertions). Based on Def. 2, we have $\Delta N =$

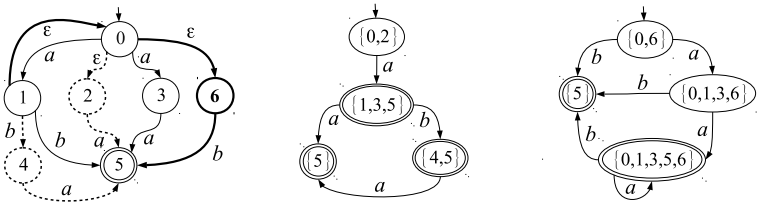

Fig. 2. (Left) a mutated NFA $\mu(\mathcal{N})$, where $\mathcal{N}$ is represented by plain and dashed lines, while μ is represented by dashed lines (removals) and bold lines (insertions). (Center) the DFA $\mathcal{D}$ *SC*-equivalent to $\mathcal{N}$. (Right) the DFA $\mathcal{D}'$ *SC*-equivalent to $\mu(\mathcal{N})$.

$\{-2, -4, +6\}$, $\Delta F = \emptyset$, $\Delta T = \{-(0 \xrightarrow{\varepsilon} 2), -(2 \xrightarrow{a} 5), -(1 \xrightarrow{b} 4), -(4 \xrightarrow{a} 5), +(0 \xrightarrow{\varepsilon} 6), +(6 \xrightarrow{b} 5), +(1 \xrightarrow{\varepsilon} 0)\}$. Centered in Fig. 2 is the DFA $\mathcal{D}$ *SC*-equivalent to $\mathcal{N}$. Based on Def. 3, the problem is generating the DFA $\mathcal{D}'$ *SC*-equivalent to $\mu(\mathcal{N})$, displayed on the right side of Fig. 2, based on $\mathcal{N}$, $\mathcal{D}$, and μ.

4. Subset Restructuring

In order to solve a mutating determinization problem (Def. 3), rather than applying *Subset Construction* to the NFA $\mu(\mathcal{N})$ out of its context, thereby disregarding $\mathcal{D}$, $\mathcal{D}'$ is determined via *Subset Restructuring*, a conservative algorithm that updates (in fact, restructures) $\mathcal{D}$ based on $\mathcal{N}$ and μ, rather than creating it from scratch. This algorithm distinguishes between the *identifier* of a DFA state, namely d, and its *extension*, denoted $\|d\|$, the latter being the set of NFA states included in d. The extension of a DFA state may change during processing, while its identifier cannot. *Subset Restructuring* makes use of a sequence $\mathcal{B}$ of *buds* (see Def. 4), which somewhat surrogates the stack $\mathcal{S}$ of states in *Subset Construction* (see Algorithm 1).

Definition 4. *Let μ be a mutation of $\mathcal{N}$, d a state of the automaton $\mathcal{D}$ being processed by* Subset Restructuring*, ℓ either a symbol of the alphabet or ε, and $\mathbb{N}$ the ℓ-closure of $\|d\|$ in $\mu(\mathcal{N})$. The triple $(d, \ell, \mathbb{N})$ is an* ℓ-bud *for $\mathcal{D}$.*

A bud $(d, \ell, \mathbb{N})$ is indicative of the need for updating the transition function of the state d, just as a state in the stack $\mathcal{S}$ of *Subset Construction* needs to be defined in its transition function. Specifically, when $\ell \neq \varepsilon$, the pair (d, ℓ) maps to a state with extension $\mathbb{N}$. *Subset Restructuring* is required to keep each state d in $\mathcal{D}$ still reachable from the initial state d_0. To this end, d is qualified by its *distance*, written $\delta(d)$, this being the minimum number of transitions connecting d_0 with d. In particular, $\delta(d_0) = 0$. Based on the distance, the set of states in $\mathcal{D}$ is partitioned into a finite set of *strata*, namely $D_0, D_1, \ldots, D_k$, with each stratum D_i including the set of states d such that $\delta(d) = i$. In particular, $D_0 = \{d_0\}$. The buds $(d, \ell, \mathbb{N})$ in the bud sequence $\mathcal{B}$ are partially ordered based on $\delta(d)$. Consequently, *Subset Restructuring* restructures $\mathcal{D}$ top-down, stratum by stratum. The stratum under processing, called the *front stratum*, is denoted $D_{\hat{\delta}}$, where $\hat{\delta}$ is the *front distance*. While being processed, $\mathcal{D}$ is partitioned into three regions: (1) the *prefix* of $\mathcal{D}$ (already processed), including strata D_i such that $i < \hat{\delta}$, (2) the front stratum $D_{\hat{\delta}}$ (under processing), and (3) the *suffix* of $\mathcal{D}$ (not yet processed), including strata D_j such that $j > \hat{\delta}$. States in the prefix are complete (in both extension and transition function) and cannot be changed by subsequent processing. By contrast, states in the suffix can subsequently change (in either extension or transition function). States in the front stratum are fixed, in both number and extension, while their transition function may change. Since states in $\mathcal{D}$ are marked by their distance, updating $\mathcal{D}$ may require distance relocation. *Subset Restructuring* performs distance relocation lazily, precisely up to the stratum $D_{\hat{\delta}+1}$. Only when the bud sequence becomes empty is distance relocation completed on the suffix of $\mathcal{D}$. To this end, a *relocation sequence* $\mathbb{R}$ is used, where states with possible changed distance are inserted.

We now present the pseudocode of *Subset Restructuring* which, for space reasons, is split into two parts, namely Algorithm 2 (lines 1–29) and Algorithm 3 (lines 30–54). For the sake of simplicity and without loss of generality, the processing of mutated final states is omitted. First, in line 5, the NFA $\mathcal{N}$ is updated based on the mutation μ. Then, in lines 6–11, the main data structures of the algorithm are initialized, namely the bud sequence $\mathcal{B}$, the front distance $\hat{\delta}$, and the relocation sequence $\mathbb{R}$. $\bar{\mathbb{N}}$ includes the states of $\mathcal{N}$ which are exited by transitions in ΔT. Intuitively, the states in $\mathcal{D}$ whose extension intersect $\bar{\mathbb{N}}$ are the first states on which the algorithm starts processing $\mathcal{D}$. To this end, $\mathcal{B}$ is assigned the aggregation of the sets of buds $\mathcal{B}_{d_0}$, $\mathcal{B}_{\varepsilon}$, and $\mathcal{B}_{\ell}$. The set $\mathcal{B}_{d_0}$ is either empty or composed of just

Algorithm 2 *Subset Restructuring* (Part 1)

1: **procedure** SUBSET RESTRUCTURING($\mathcal{N}$, $\mathcal{D}$, μ)
2: $\mathcal{N}$: an NFA
3: $\mathcal{D} = (D, \Sigma, T_d, d_0, F_d)$: the DFA SC-equivalent to $\mathcal{N}$
4: $\mu = (\Delta N, \Delta T, \Delta F)$: a mutation of $\mathcal{N}$

5: Update $\mathcal{N}$ by μ, thereby obtaining $\mu(\mathcal{N})$, with N' being the set of states
6: $\bar{\mathbb{N}} \leftarrow \{ n \mid n \in (N' \setminus \Delta N), n \xrightarrow{\ell} n' \in \Delta T\}$
7: $\mathcal{B}_{d_0} \leftarrow \{(d_0, \varepsilon, \mathbb{N}) \mid n_0 \xrightarrow{\varepsilon} n' \in \Delta T, \mathbb{N} = \varepsilon\text{-closure}(n_0), \mathbb{N} \neq \|d_0\|\}$
8: $\mathcal{B}_\varepsilon \leftarrow \{(d, \ell, \mathbb{N}) \mid d \xrightarrow{\ell} d' \in T_d, n \in \|d'\| \cap \bar{\mathbb{N}}, n \xrightarrow{\varepsilon} n' \in \Delta T, \mathbb{N} = \ell\text{-closure}(\|d\|)\}$
9: $\mathcal{B}_\ell \leftarrow \{(d, \ell, \mathbb{N}) \mid \ell \neq \varepsilon, d \in D, n \in \|d\| \cap \bar{\mathbb{N}}, n \xrightarrow{\ell} n' \in \Delta T, \mathbb{N} = \ell\text{-closure}(\|d\|)\}$
10: $\mathcal{B} \leftarrow \mathcal{B}_{d_0} \cup \mathcal{B}_\varepsilon \cup \mathcal{B}_\ell$
11: $\mathbb{R} \leftarrow [\,], \quad \hat{\delta} \leftarrow 0$
12: **while** $\mathcal{B}$ is not empty **do**
13: Let $\bar{d}$ be the state relevant to the first bud of $\mathcal{B}$
14: **if** $\delta(\bar{d}) > \hat{\delta}$ **then**
15: PROPAGATE($\mathbb{R}$) ⟨ *Propagation is up to stratum* $D_{\delta(\bar{d})+1}$ ⟩
16: **end if**
17: Remove the first bud $(d, \ell, \mathbb{N})$ from $\mathcal{B}$
18: $\hat{\delta} \leftarrow \delta(d)$
19: **if** $\ell = \varepsilon$ **then** ⟨ *Rule* $\mathcal{R}_0$ ⟩
20: UPDATE($d, \mathbb{N}$) ⟨ *d is necessarily the initial state* d_0 ⟩
21: **else if** no ℓ-transition exits d **then**
22: **if** $d' \in D, \|d'\| = \mathbb{N}$ **then** ⟨ *Rule* $\mathcal{R}_1$ ⟩
23: Insert transition $d \xrightarrow{\ell} d'$ into $\mathcal{D}$
24: RELOCATE($d', \delta(d) + 1$)
25: **else** ⟨ *Rule* $\mathcal{R}_2$ ⟩
26: Create a new state d' in D, where $\|d'\| = \mathbb{N}$, along with its buds
27: Insert transition $d \xrightarrow{\ell} d'$ into $\mathcal{D}$
28: $\delta(d') \leftarrow \delta(d) + 1$
29: **end if** ⟨ *Part 2 of* Subset Restructuring *is continued in Algorithm 3* ⟩

one ε-bud pertaining to the initial state d_0. Instead, $\mathcal{B}_\varepsilon$ includes buds $(d, \ell, \mathbb{N})$ such that there is an ℓ-transition in $\mathcal{D}$ exiting d and entering d', where the extension of d' contains a state of $\bar{\mathbb{N}}$ that is exited by an ε-transition of μ. Finally, $\mathcal{B}_\ell$ includes the ℓ-buds relevant to the states d of $\mathcal{D}$ whose extension intersects $\bar{\mathbb{N}}$ and are exited by an ℓ-transition of μ. What buds in $\mathcal{B}_\varepsilon$ and $\mathcal{B}_\ell$ have in common is that the transition function of the corresponding state d is possibly changed owing to the mutation. In other words, the initial buds assigned to $\mathcal{B}$ in line 10 denote the root states in $\mathcal{D}$ in which *Subset Restructuring* starts its processing. Then, one bud at a time is removed from $\mathcal{B}$ and processed in the main loop (lines 12–49). In lines 13–16, before actually processing the bud, a check is performed on the distance of the involved state $\bar{d}$. Specifically, if $\bar{d}$ belongs to the suffix of $\mathcal{D}$, then the processing of the front stratum $D_{\hat{\delta}}$ is complete; hence, distance propagation is carried out by the auxiliary procedure *Propagate*. After removing the first bud $(d, \ell, \mathbb{N})$ from $\mathcal{B}$ and setting the front distance $\hat{\delta}$ (lines 17 and 18), the bud is processed based on the symbol ℓ and the current transitions exiting d. Seven processing rules come into play, namely $\mathcal{R}_0, \ldots, \mathcal{R}_6$.

In rule $\mathcal{R}_0$ (lines 19 and 20), when $\ell = \varepsilon$, $\|d\|$ is replaced by $\mathbb{N}$. Based on line 7, d is necessarily the initial state d_0. In rule $\mathcal{R}_1$ (lines 22–24), when d is not exited by any ℓ-transition and there is a state d' such that $\|d'\| = \mathbb{N}$, a transition $d \xrightarrow{\ell} d'$ is created, with the distance of d' being possibly updated by the auxiliary procedure *Relocate*. In rule $\mathcal{R}_2$ (lines 26–28), when d is not exited by any ℓ-transition and there is no state d' such that $\|d'\| = \mathbb{N}$, both a new state d', with $\|d'\| = \mathbb{N}$, and a new transition $d \xrightarrow{\ell} d'$ are created. To generate the transition function of d' subsequently,

Algorithm 3 *Subset Restructuring* (Part 2)

30: **else** ⟨ *At least one ℓ-transition exits d* ⟩
31: **for all** $t \in T_d, t = d \xrightarrow{\ell} d', \|d'\| \neq \mathbb{N}$ **do**
32: **if** $d' \neq d_0$, no other trans. enters d' **then** ⟨ *Rule* $\mathcal{R}_3$ ⟩
33: UPDATE$(d', \mathbb{N})$
34: **else if** $d' = d_0$ **or** $t' \in T_d, t' \neq t, t' = d_p \xrightarrow{x} d', \delta(d_p) \leq \hat{\delta}$ **then**
35: **if** $d'' \in D, \|d''\| = \mathbb{N}$ **then** ⟨ *Rule* $\mathcal{R}_4$ ⟩
36: Redirect t toward d''
37: RELOCATE$(d'', \delta(d) + 1)$
38: **else** ⟨ *Rule* $\mathcal{R}_5$ ⟩
39: Create a new state d'' in D, where $\|d''\| = \mathbb{N}$, along with its buds
40: Redirect t toward d''
41: $\delta(d'') \leftarrow \delta(d) + 1$
42: **end if**
43: **else** ⟨ *Rule* $\mathcal{R}_6$ ⟩
44: Remove the transitions entering d' other than t and replace them with buds
45: UPDATE$(d', \mathbb{N})$
46: **end if**
47: **end for**
48: **end if**
49: **end while**
50: **if** $d \in D, \|d\| = \emptyset$ **then**
51: Remove d and all its entering/exiting transitions
52: **end if**
53: PROPAGATE$(\mathbb{R})$ ⟨ *Distance propagation is applied to states in the suffix of $\mathcal{D}$* ⟩
54: **end procedure**

the buds for d' are inserted into $\mathcal{B}$ based on the transition function of the NFA states in $\|d'\|$. Specifically, the set of buds generated for d' in line 26 is $\{(d', \ell', \mathbb{N}') \mid n \in \|d'\|, n \xrightarrow{\ell'} n' \in \mathcal{N}', \mathbb{N}' = \ell'\text{-closure}(\|d'\|)\}$.

Rules $\mathcal{R}_3, \ldots, \mathcal{R}_6$ (involved in Part 2, Algorithm 3), occur when there is at least one ℓ-transition exiting d. Note that, owing to possible merging of states by *Update*, several ℓ-transitions may exit the same state in $\mathcal{D}$. Still, this nondeterminism in $\mathcal{D}$ will disappear before ending the processing of buds. Hence, each of these rules can be applied several times for the same bud, as specified by the loop in lines 31–47. In rule $\mathcal{R}_3$ (lines 32 and 33), when d' is not the initial state and no other transition enters d', the extension of d' is updated by the auxiliary procedure *Update*. In rule $\mathcal{R}_4$ (lines 35–37), when d' is the initial state or there is another transition entering d' from a state d_p such that $\delta(d_p) \leq \hat{\delta}$ and there is d'' such that $\|d''\| = \mathbb{N}$, the transition exiting d is redirected toward d'', whose distance needs relocation. Rule $\mathcal{R}_5$ (lines 39–41) is applied based on the same conditions of $\mathcal{R}_4$, except that there is no state d'' such that $\|d''\| = \mathbb{N}$. If so, a new state d'' is created (along with relevant buds), with $\|d''\| = \mathbb{N}$, and t is redirected toward d''. Rule $\mathcal{R}_6$ (lines 44 and 45) occurs when the condition in line 34 is not fulfilled. If so, then all other transitions entering d' are removed and replaced by buds, while the extension of d' is updated based on $\mathbb{N}$. When the condition in line 34 holds, the reachability of d' from the initial state does not depend on t, which thereby can be redirected toward another state (rules $\mathcal{R}_4$ and $\mathcal{R}_5$). By contrast, when this condition does not hold, redirecting t away from d' may result in the disconnection of d'. This is why in $\mathcal{R}_6$ all other transitions entering d' are removed (replaced by buds), while preserving $d \xrightarrow{\ell} d'$. In line 51, when $\mathcal{B}$ becomes empty, if $\mathcal{D}$ includes an empty state d, then d and all the transitions entering/exiting d are removed from $\mathcal{D}$. Note that retaining the empty state until $\mathcal{B} = \emptyset$ is essential in order to avoid the disconnection of $\mathcal{D}$. Eventually, distance relocation is propagated on the suffix of $\mathcal{D}$ (line 53).

The processing state of *Subset Restructuring* is called a *configuration*, namely a pair $(\bar{\mathcal{D}}, \bar{\mathcal{B}})$, where $\bar{\mathcal{D}}$ is the current instance of the automaton $\mathcal{D}$, and $\bar{\mathcal{B}}$ the current instance of the bud sequence $\mathcal{B}$. During processing, the algorithm performs a *trajectory*, namely a finite sequence $[\alpha_0, \alpha_1, \ldots, \alpha_q]$ of configurations, where $\alpha_0 = (\mathcal{D}_0, \mathcal{B}_0)$ is the initial

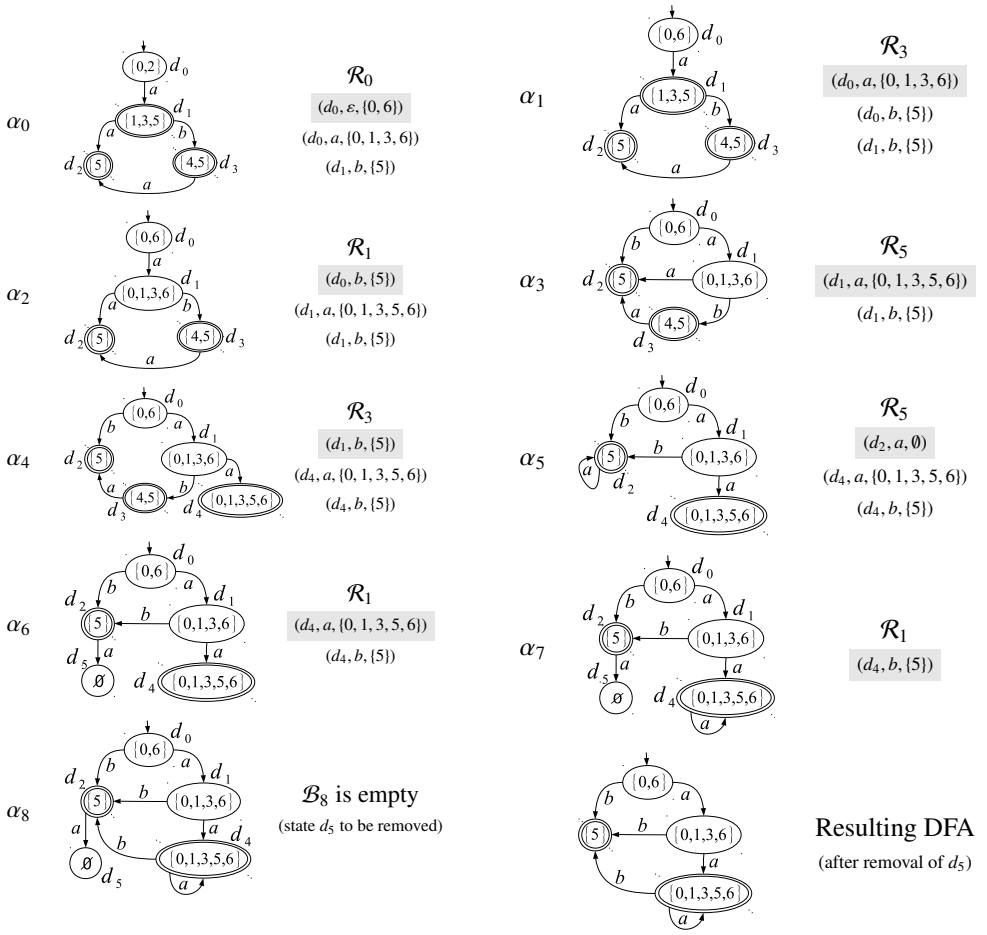

Fig. 3. Trajectory of *Subset Restructuring* in Example 5. For each configuration $\alpha_i = (\mathcal{D}_i, \mathcal{B}_i)$, $i \in [0\,..\,8]$, the bud processed is highlighted and the processing rule is written on the top. The DFA resulting after the removal of the empty state is outlined on the bottom-right side (cf. Fig. 2).

configuration, with $\mathcal{D}_0$ being the DFA *SC*-equivalent to $\mathcal{N}$ and $\mathcal{B}_0$ is the initial instance of $\mathcal{B}$ (line 10, Algorithm 2), while $\alpha_q = (\mathcal{D}_q, \mathcal{B}_q)$ is the final configuration, with $\mathcal{D}_q$ being the DFA *SC*-equivalent to $\mu(\mathcal{N})$ and $\mathcal{B}_q$ being empty.

Example 5. Consider the mutation determinization problem defined in Example 4. The trajectory of *Subset Restructuring* in determinizing $\mu(\mathcal{N})$ is traced in Fig. 3. For each configuration $\alpha_i = (\mathcal{D}_i, \mathcal{B}_i), i \in [0\,..\,8]$, the bud processed is highlighted and the corresponding processing rule is written on top of it. To highlight the strata, the states of $\mathcal{D}_i$ belonging to the same stratum are aligned horizontally. For instance, considering the instance $\mathcal{D}_0$ of the configuration α_0, we have three strata: $D_0 = \{d_0\}$, $D_1 = \{d_1\}$, and $D_2 = \{d_2, d_3\}$. According to the initialization of $\mathcal{B}$ in lines 6–10 of Algorithm 2, we have $N' \setminus \Delta N = \{0, 1, 3, 5\}$ and $\bar{\mathbb{N}} = \{0, 1\}$. Also, $\mathcal{B}_{d_0} = \{(d_0, \varepsilon, \{0, 6\})\}$, $\mathcal{B}_\varepsilon = \{(d_0, a, \{0, 1, 3, 6\})\}$, and $\mathcal{B}_\ell = \{(d_1, b, \{5\})\}$. Hence, based on line 10, $\mathcal{B}$ is initialized with these three singletons, which corresponds to instance $\mathcal{B}_0$ of configuration α_0 in Fig. 3. Not coincidentally, the final automaton (bottom-right of Fig. 3) equals the DFA $\mathcal{D}'$ which is *SC*-equivalent to $\mu(\mathcal{N})$ (displayed on the right side of Fig. 2).

5. Correctness and experimentation

Subset Restructuring is sound and complete, in other words, it generates the same DFA as that generated by *Subset Construction* from scratch. Both *Subset Construction* and *Subset Restructuring* have been implemented in C++ and compared in the determinizatinon of a large set of MFAs[23]. Specifically, hundreds of thousands of problem instances

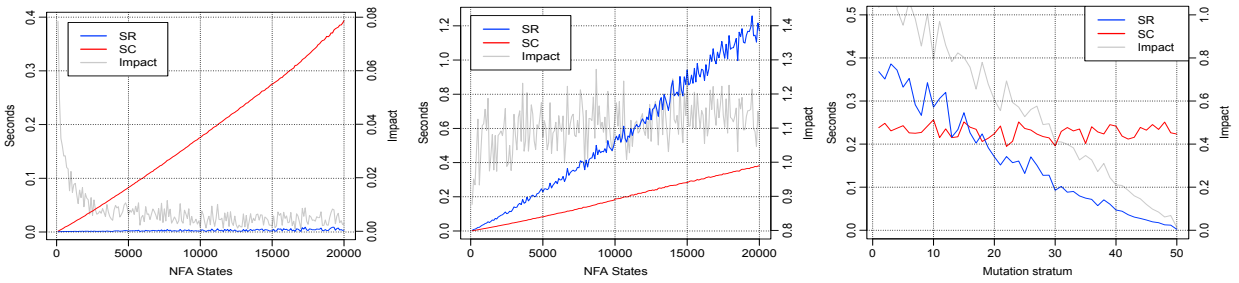

Fig. 4. Experimental results.

generated randomly have confirmed the correctness of *Subset Restructuring* empirically. They have also indicated that, when the NFA is large and the mutations are small, *Subset Restructuring* outperforms *Subset Construction* in a vast majority of cases, but not always. The reason for this can be grasped based on the notions of *impact* and *gain*.

Definition 5. *Let $\mathcal{D}$ be the DFA* SC*-equivalent to an NFA $\mathcal{N}$, μ a mutation of $\mathcal{N}$, and $\mathcal{D}'$ the DFA* SC*-equivalent to $\mu(\mathcal{N})$. Let n be the number of states in $\mathcal{D}$, let n' the number of states in $\mathcal{D}'$, and let n_r the number of states either created, deleted, or otherwise processed by* Subset Restructuring *in generating $\mathcal{D}'$ (based on $\mathcal{D}$ and μ). The* impact *of restructuring $\mathcal{D}$ is the ratio of n_r to n', namely $\Im = n_r/n'$.*

Intuitively, the smaller the impact, the better the performance of *Subset Restructuring* over *Subset Construction* will be. Given a mutating determinization problem, what is the possible range of the impact? In the best case, only one state is processed by *Subset Restructuring*; hence, $\Im = 1/n'$. In the worst case, $\mathcal{D}'$ is composed on the initial state only; hence, $\Im = n'$, as $n - 1$ states are removed from $\mathcal{D}$ and the initial state is processed. In summary, we have $1/n' \leq \Im \leq n'$. With reference to Def. 5, we define the *gain*, namely γ, as follows:

$$\gamma = \begin{cases} (n' - n_r)/n' & \text{if } n' \geq n_r \\ (n' - n_r)/n_r & \text{otherwise} \end{cases} = \begin{cases} 1 - \Im & \text{if } \Im \leq 1 \\ (1/\Im) - 1 & \text{otherwise.} \end{cases} \tag{1}$$

Based on eqn. (1), we have: $-1 < \gamma < 1$. When positive, γ indicates the fraction of states unprocessed by *Subset Restructuring* against *Subset Construction*. When negative, $|\gamma|$ indicates the fraction of states unprocessed by *Subset Construction* against *Subset Restructuring*. When $\gamma = 0$, the two algorithms process the same number of states. Hence, in theory, only when $\Im < 1$ is *Subset Restructuring* expected to be more convenient than *Subset Construction*. But, is this expectation confirmed by the experimentation? To perform experiments, a specific auxiliary algorithm has been designed, which generates pseudo-random instances of the mutating determinization problem based on a set of parameters, namely: n, the number of states of $\mathcal{N}$; σ, the number of labels in Σ; δn, the number of states either inserted into or removed from $\mathcal{N}$; δt, the number of transitions mutated; β, the *branching factor*, which is the average number of transitions exiting each state in $\mathcal{N}$; and $\mathcal{E}$, the percentage of ε-transitions in $\mathcal{N}$.

Experimentation was performed on large MFAs with relatively small mutations. All parameters were kept constant, with the exception of n, specifically: $\sigma = 15$, $\delta t = 10$, $\beta = 2$, $\mathcal{E} = 0.01$, and $\delta n = 0$. Notice that $\delta n = 0$ indicates that no state is inserted or removed deliberately; still, all the states which become unreachable owing to the removal of transitions will be inserted into the mutation implicitly. Parameter n was varied from 100 to 20000 in steps of 100, hence $n = 100, 200, \ldots, 20000$. For each single value of n, about 250 instances of the problem were generated and solved by both *Subset Construction* and *Subset Restructuring* (hence, about 50000 instances in total). Resulting figures indicate that, considering the processing time, *Subset Restructuring* outperforms *Subset Construction* in the 84% of the experiments, as indicated by the plot shown on the left side of Fig. 4, while the results relevant to the remaining 16% of the experiments, where *Subset Restructuring* is outperformed by *Subset Construction*, are displayed by the plot on the center of the figure. In both plots, the x-axis indicates the number of states in the NFA $\mathcal{N}$, the y-axis on the left side indicates the processing time (in seconds), and the y-axis on the right side indicates the impact $\Im$ of restructuring the DFA $\mathcal{D}$. The processing time of *Subset Restructuring* (SR) and *Subset Construction* (SC) is represented by the blue line and the red line, respectively, where each time value is the average of about 250 different experiments generated randomly based on the same values of the parameters, in particular, the same number of NFA states. Finally, the gray

line indicates the value of the impact of the restructuring (y-axis on the right side). Not unexpectedly, the impact has a direct influence on the performance of *Subset Restructuring*, as the tests where *Subset Restructuring* performs poorly against *Subset Construction* are those with high impact. This means that, despite the mutation being small, a large number of states may be created and/or deleted in $\mathcal{D}$.

In order to have more control on the impact in the experiments, a special class of *stratified* NFAs was designed. In a stratified NFA, the set of states is partitioned into a sequence $[N_0, N_1, \ldots, N_{\delta_{\max}}]$ of *strata*, where each stratum N_i, $i \in [0\,..\,\delta_{\max}]$, includes the NFA states at distance i (from the initial state). With the exception of N_0, which includes just the initial state, each stratum contains about the same number of states. Each state in N_i is entered by one or more transitions exiting states in N_{i-1} only. Each transition exiting a state in N_i enters a state which is either in N_i or in N_{i+1}. A mutation μ in the NFA involves the removal of one state and its transitions, which possibly requires the insertion of new transitions in order to keep the child states still connected with the initial state. Hence, transitions exiting a state in N_i may not enter a state in N_j if $j < i$. Based on these rules, several instances of the mutating determinization problem were generated with $\delta_{\max} = 50$. Shown on the right side of Fig. 4 is the relation among the stratum in which the mutation μ occurs, the processing time of *Subset Construction* and *Subset Restructuring* (y-axis on the left side), and the impact $\mathfrak{I}$ (y-axis on the right side). Each time value is the average of about 100 experiments relevant to the same stratum, with a total number of 5152 experiments. As expected, the more distant the stratum from the initial state, the smaller the impact and, as a consequence, the smaller the processing time of *Subset Restructuring*. By contrast and unsurprisingly, the processing time of *Subset Construction* remains practically constant.

6. Application domain: model-based diagnosis of active systems

Historically, the need for a conservative algorithm like *Subset Restructuring* originated in a specific field of artificial intelligence called model-based diagnosis of active systems[3]. Once an active system $\mathcal{A}$ (a sort of discrete-event system) is modeled as a network of communicating automata, we can diagnose $\mathcal{A}$ based on a temporal observation[24], which is a directed acyclic graph whose nodes are marked by observable labels and arcs denote temporal precedences between these labels. Based on a temporal observation O of $\mathcal{A}$, the diagnosis engine reconstructs the behavior of $\mathcal{A}$ which is consistent with O. This behavior is a DFA whose nodes are pairs $(S, \mathfrak{I})$, where S is state of $\mathcal{A}$ and $\mathfrak{I}$ is an index of O, while arcs are marked by transitions of the communicating automata embodied in $\mathcal{A}$ (its components). An index of O is the state of a DFA, called the index space of O and denoted $\mathcal{D}(O)$, derived from O through an NFA called the nondeterministic index space of O and denoted $\mathcal{N}(O)$. The important point is that $\mathcal{D}(O)$ is SC-equivalent to $\mathcal{N}(O)$. If O is given once and for all, then $\mathcal{D}(O)$ can be generated from $\mathcal{N}(O)$ by *Subset Construction* without any problem. If, instead, O changes over time (for example by additional nodes and arcs), the diagnosis engine is required to update both $\mathcal{N}(O)$ and $\mathcal{D}(O)$ dynamically. If $\mathcal{N}(O)$ is large, then generating the new $\mathcal{D}(O)$ by *Subset Construction* is in general less than optimal, as it requires the complete construction of the DFA. This is why it is more convenient to update $\mathcal{D}(O)$ (rather than building it from scratch) based on the mutation occurred in $\mathcal{N}(O)$ as a consequence of the mutated temporal observation O. In so doing, *Subset Restructuring* may perform better than *Subset Construction*. Experimental results in monitoring-based diagnosis of active systems have confirmed the viability of this approach.

7. Conclusion

In real life, the convenience in restructuring a house instead of rebuilding it from scratch largely depends on the extent of the restructuring task. Provided that the result will be the same (correctness), the smaller the extent of the restructuring, the more convenient the restructuring against the complete rebuilding of the house will be. This simple idea has been adopted in this paper for coping with mutating determinization problems. As the experimental results suggest, the convenience in restructuring a DFA (based on a mutation of the equivalent NFA) rather than building the new DFA from scratch largely depends on the impact of the restructuring: the smaller the impact, the more convenient *Subset Restructuring* against *Subset Construction* will be in the determinization of the mutated NFA. Somewhat surprisingly, the experiments also indicate that even a small mutation of the NFA may result in a large impact of the restructuring. This counterintuitive fact can be grasped by considering that each state of the DFA is identified by a subset of the states of the equivalent NFA. Hence, if we insert/remove a few states into/from the NFA, then, in the worst case, all of the states of the DFA may be extended/shrunk by some of these NFA states. In other

words, even a small mutation may result in a large impact. Thus, whether to adopt *Subset Restructuring* rather than *Subset Construction* remains an empirical question. In practice, given an application domain with specific mutating determinization problems to be solved, the choice between the two algorithms may be determined based on real test cases aimed at pinpointing the trend of the impact of the restructuring in the specific domain.

References

1. Rabin, M., Scott, D.. Finite automata and their decision problems. *IBM Journal of Research and Development* 1959;**3**(2):114–125. doi:10.1147/rd.32.0114.
2. Hopcroft, J., Motwani, R., Ullman, J.. *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley; third ed.; 2006.
3. Lamperti, G., Zanella, M.. *Diagnosis of Active Systems: Principles and Techniques*; vol. 741 of *Springer International Series in Engineering and Computer Science*. Dordrecht, Netherlands: Springer; 2003. doi:10.1007/978-94-017-0257-7.
4. Lamperti, G., Zanella, M.. Monitoring of active systems with stratified uncertain observations. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans* 2011;**41**(2):356–369. doi:10.1109/TSMCA.2010.2069096.
5. Lamperti, G., Scandale, M.. From diagnosis of active systems to incremental determinization of finite acyclic automata. *AI Communications* 2013;**26**(4):373–393. doi:10.3233/AIC-130574.
6. Lamperti, G., Zhao, X.. Diagnosis of active systems by semantic patterns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 2014;**44**(8):1028–1043. doi:10.1109/TSMC.2013.2296277.
7. Lamperti, G., Quarenghi, G.. Intelligent monitoring of complex discrete-event systems. In: Czarnowski, I., Caballero, A., Howlett, R., Jain, L., editors. *Intelligent Decision Technologies 2016*; vol. 56 of *Smart Innovation, Systems and Technologies*. Springer International Publishing Switzerland; 2016, p. 215–229. doi:10.1007/978-3-319-39630-9_18.
8. Aichernig, B., Jöbstl, E., Kegele, M.. Incremental refinement checking for test case generation. In: Veanes, M., Viganò, L., editors. *7th International Conference on Tests and Proofs (TAP 2013)*; vol. 7942 of *Lecture Notes in Computer Science*. Budapest: Springer, Berlin, Heidelberg. ISBN 3-540-19074-0; 2013, p. 1–19. doi:10.1007/978-3-642-38916-0_1.
9. Aichernig, B., Jöbstl, E., Tiran, S.. Model-based mutation testing via symbolic refinement checking. *Science of Computer Programming* 2015;**97**(4):383–404. doi:10.1016/j.scico.2014.05.004.
10. Lamperti, G., Zanella, M., Chiodi, G., Chiodi, L.. Incremental determinization of finite automata in model-based diagnosis of active systems. In: Lovrek, I., Howlett, R., Jain, L., editors. *Knowledge-Based Intelligent Information and Engineering Systems*; vol. 5177 of *Lecture Notes in Artificial Intelligence*. Springer, Berlin, Heidelberg. ISBN 978-3-540-85562-0; 2008, p. 362–374.
11. Balan, S., Lamperti, G., Scandale, M.. Incremental subset construction revisited. In: Neves-Silva, R., Tshirintzis, G., Uskov, V., Howlett, R., Jain, L., editors. *Smart Digital Futures*; vol. 262 of *Frontiers in Artificial Intelligence and Applications*. Amsterdam: IOS Press; 2014, p. 25–37.
12. Balan, S., Lamperti, G., Scandale, M.. Metrics-based incremental determinization of finite automata. In: Teufel, S., Tjoa, A.M., You, I., Weippl, E., editors. *Availability, Reliability, and Security in Information Systems*; vol. 8708 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg. ISBN 978-3-319-10974-9; 2014, p. 29–44.
13. Lamperti, G., Scandale, M., Zanella, M.. Determinization and minimization of finite acyclic automata by incremental techniques. *Software: Practice and Experience* 2016;**46**(4):513–549. doi:10.1002/spe.2309.
14. Brognoli, S., Lamperti, G., Scandale, M.. Incremental determinization of expanding automata. *The Computer Journal* 2016;**59**(12):1872–1899. doi:10.1093/comjnl/bxw044.
15. Lamperti, G., Zhao, X.. Decremental subset construction. In: Czarnowski, I., Howlett, R., Jain, L., editors. *Intelligent Decision Technologies 2017*; vol. 72 of *Smart Innovation, Systems and Technologies*. Springer International Publishing. ISBN 978-3-319-59420-0; 2018, p. 22–36. doi:10.1007/978-3-319-59421-7_3.
16. Daciuk, J., Mihov, S., Watson, B., Watson, R.. Incremental construction of minimal acyclic finite state automata. *Computational Linguistics* 2000;**26**(1):3–16. doi:10.1162/089120100561601.
17. Carrasco, R., Forcada, M.. Incremental construction and maintenance of minimal finite-state automata. *Computational Linguistics* 2002;**28**(2):207–216. doi:10.1162/089120102760173652.
18. Watson, B., Daciuk, J.. An efficient incremental DFA minimization algorithm. *Natural Language Engineering* 2003;**9**(1):49–64. doi:10.1017/S1351324903003127.
19. Daciuk, J.. Semi-incremental addition of strings to a cyclic finite automaton. In: Klopotek, M., Wierzchon, S., Trojanowski, K., editors. *Intelligent Information Processing and Web Mining*; vol. 25 of *Advances in Soft Computing*. Springer, Berlin, Heidelberg; 2004, p. 201–207. doi:10.1007/978-3-540-39985-8_21.
20. Champarnaud, J.M., Coulon, F., Paranthoën, T.. Brute force determinization of NFAs by means of state covers. *International Journal of Foundations of Computer Science* 2005;**16**(3):441–451. doi:10.1142/S012905410500308X.
21. Carrasco, R., Daciuk, J., Forcada, M.. Incremental construction of minimal tree automata. *Algorithmica* 2009;**55**(1):95–110. doi:10.1007/s00453-008-9172-4.
22. Aho, A., Lam, M., Sethi, R., Ullman, J.. *Compilers – Principles, Techniques, and Tools*. Reading, MA: Addison-Wesley; second ed.; 2006.
23. Caniato, G.. *Algoritmi per la determinizzazione di automi mutanti*. Master's thesis; Department of Information Enginneering, University of Brescia, Italy; 2018.
24. Lamperti, G., Zanella, M.. Diagnosis of discrete-event systems from uncertain temporal observations. *Artificial Intelligence* 2002;**137**(1–2):91–163. doi:10.1016/S0004-3702(02)00123-6.