On Complementation of Nondeterministic Finite Automata without Full Determinization

Lukáš Holík[1,2], Ondřej Lengál[1],
Juraj Major[3], Adéla Štěpková[3], and Jan Strejček[3]

[1] Brno University of Technology, Brno, Czech Republic
[2] Aalborg University, Aalborg, Denmark
[3] Masaryk University, Brno, Czech Republic

Abstract. Complementation of finite automata is a basic operation used in numerous applications. The standard way to complement a nondeterministic finite automaton (NFA) is to transform it into an equivalent deterministic finite automaton (DFA) and complement the DFA. The DFA can, however, be exponentially larger than the corresponding NFA. In this paper, we study several alternative approaches to complementation, which are based either on reverse powerset construction or on two novel constructions that exploit a commonly occurring structure of NFAs. Our experiment on a large data set shows that using a different than the classical approach can, in many cases, yield significantly smaller complements.

1 Introduction

Complementation of *finite automata* is an operation with many applications in formal methods. It is used, e.g., in regular model checking [25,6,5], representing extended regular expressions [31,10], to implement negation in automata-based decision procedures for logics such as Presburger arithmetic [33,17] or monadic second order theories like WS1S or MSO(Str) [7,11,16,24,14,13,2], or as the basic underlying operation for testing language inclusion and equivalence over automata. Complementing *deterministic finite automata* (DFAs) is an easy task: it is sufficient to add a single state, direct all missing transitions to this state, and swap accepting and non-accepting states.

In practice, *nondeterministic finite automata* (NFAs) are often favored over DFAs due to their potentially much (up to exponentially) smaller size. The classical approach to complementing NFAs goes through determinization of the input NFA using the *powerset construction* into a DFA and then using DFA complementation. While easy to implement, this approach is prone to cause a blow-up in the number of states, as determinizing an NFA with n states may, in the worst case, result in a DFA with 2^n states [28] and the size of the complement would then also be exponential. Some automata-based algorithms are highly sensitive to the sizes of complement automata, such as decision procedures of certain logics [33,17,7,11,16,24,14,13,2], where the output of the complement may be the basic structure over which another complementation is performed (usually after projection, which can turn a potentially deterministic automaton into a nondeterministic one); for some of the logics, the increase in the size of the complement is the underlying cause of their non-elementary complexity [27]. Due to this, some of the applications tried to avoid complementation altogether, for instance using symbolic techniques [31,14,13]. This is, however, not always possible or feasible, as symbolic techniques often disallow to use, e.g., standard automata reduction techniques (cf., [8,22]), which often have a great impact on the performance of the applications.

Since the lower bounds on the worst-case sizes of deterministic and complement automata are both 2^n [29,3,20,23], one might think that the determinization-based approach is optimal. In practice, this is, however, far from the truth. Consider, e.g., the NFA $\mathcal{A}_2$ accepting the language $\{a,b\}^*.\{a\}.\{a,b\}^2$ shown in Fig. 1 (left). While the (minimal) deterministic complement has 8 states, there exists an NFA with 4 states given in Fig. 1 (right) that is a complement of $\mathcal{A}_2$. This complement was obtained as follows. We reversed $\mathcal{A}_2$, then determinized it (which is easy as $\mathcal{A}_2$ is reverse-deterministic), complemented the output by swapping accepting and non-accepting states, reversed again, and, finally, removed one unreachable state. We can generalize the example above to the family of NFAs $\mathcal{A}_n$ whose languages are $\{a,b\}^*.\{a\}.\{a,b\}^n$ for $n \in \mathbb{N}$. Here, the size of the minimal complement DFA for the NFA $\mathcal{A}_n$ is 2^{n+1}, while the complement NFA constructed by the *reverse powerset* procedure mentioned above has $n+2$ states, the same as $\mathcal{A}_n$. This example is a motivation for a deeper study of NFA complementation.

In this paper, we present several alternative approaches to NFA complementation. Besides the reverse powerset complementation (Section 3), we introduce two novel complementation constructions that target NFAs with a particular structure (containing several strongly connected components), common in practice. We first introduce a basic version of the novel constructions on NFAs with a very restricted structure (Section 4) and then we briefly present the generalized version of these constructions and the generalized complementation problem that allows to combine these constructions (Section 5). Our experimental evaluation (Section 6) shows that in a significant number of cases, using an alternative complementation method can give a much smaller complement than the classical construction. Due to the page limit, many parts of the paper are relegated to the technical report [21], including a precise description of the generalized complementation constructions with the corresponding correctness proofs, an example of NFA subclass with a subexponential complement size, implementation details, and additional experiments.

Related Work. While complementation of automata over infinite words is a lively topic (e.g., [19,1,4]), complementation of NFAs seems to be under-researched with not many relevant prior work. The powerset approach to determinization, which is the basic block of the classical complementation, can be traced to Rabin and Scott [28]. Optimizations of the powerset determinization were proposed in [15].

From the theoretical side, the exponential lower bound of NFA complementation was studied with respect to various alphabet sizes (the larger the alphabet size, the easier it is to construct an NFA whose complement is forced to be exponential) in [29,3,20,23].

2 Preliminaries

An *alphabet* Σ is a finite nonempty set of *symbols*. A *word* over Σ is a sequence $u = u_1 \dots u_n$ where $u_i \in \Sigma$ for all $1 \leq i \leq n$, with its *length* n denoted by $|u|$. The *empty word* is denoted by ε. The set of all words over Σ is denoted by Σ^*, and its subsets are *languages* over Σ. The *concatenation* of u with a word $v = u_{n+1} \dots u_m$ is the word $uv = u_1 \dots u_m$. The *reverse* of u is the word $rev(u) = u_n u_{n-1} \dots u_1$. The *concatenation* of languages $L, L' \subseteq \Sigma^*$ is the language $L.L' = \{uv \mid u \in L, v \in L'\}$. The *reverse* of L is the language $rev(L) = \{rev(w) \mid w \in L\}$ and its *complement* is the language $co(L) = \Sigma^* \setminus L$.

Finite Automata. A *nondeterministic finite automaton (NFA)* is defined as a tuple $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of *states*, Σ is an alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a *transition relation*, $I \subseteq Q$ is a set of *initial states*, and $F \subseteq Q$ is a set of *accepting states* (also called *final states*). We write $q \xrightarrow{a} r \in \delta$ instead of $(q, a, r) \in \delta$. If δ is clear from the context, we write just $q \xrightarrow{a} r$. The *size* of $\mathcal{A}$ is defined as $|\mathcal{A}| = |Q|$. We abuse the notation and use δ also as the function $\delta\colon Q \times \Sigma \to 2^Q$ defined as $\delta(q, a) = \{r \in Q \mid q \xrightarrow{a} r\}$. We also extend the function δ to sets of states $P \subseteq Q$ as $\delta(P, a) = \bigcup_{q \in P} \delta(q, a)$. A *deterministic finite automaton (DFA)* is an NFA with $|I| = 1$ and $|\delta(q, a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$. A DFA is *complete* if $|\delta(q, a)| = 1$ for all $q \in Q$ and $a \in \Sigma$. A *run* of an NFA $\mathcal{A}$ over a word $v = v_1 v_2 \dots v_n$ is a sequence of transitions $q_0 \xrightarrow{v_1} q_1, q_1 \xrightarrow{v_2} q_2, \dots, q_{n-1} \xrightarrow{v_n} q_n$ with $q_0 \in I$. It is *accepting* if $q_n \in F$. A state that appears in a run over some word $v \in \Sigma^*$ is *reachable*, else it is *unreachable*. A state is *reachable from a state* q if it is reachable in the NFA $(Q, \Sigma, \delta, \{q\}, F)$. $\mathcal{A}$ *accepts* the language $\mathcal{L}(\mathcal{A})$ of all words over Σ for which it has an accepting run. Automata $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* iff $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

The *reverse* of $\mathcal{A}$ is the NFA $rev(\mathcal{A}) = (Q, \Sigma, rev(\delta), F, I)$, where $rev(\delta) = \{r \xrightarrow{a} q \mid q \xrightarrow{a} r \in \delta\}$. $\mathcal{A}$ is called *reverse-deterministic* iff $rev(\mathcal{A})$ is a DFA. An NFA $\mathcal{C}$ is called a *complement* of $\mathcal{A}$ with respect to an alphabet Λ if $\mathcal{L}(\mathcal{C}) = \Lambda^* \setminus \mathcal{L}(\mathcal{A})$. If Λ is not specified, we assume that $\Lambda = \Sigma$. Given two NFAs $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, I_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, I_2, F_2)$ where $Q_1 \cap Q_2 = \emptyset$, their *union* is the NFA $\mathcal{A}_1 \Cup \mathcal{A}_2 = (Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2, I_1 \cup I_2, F_1 \cup F_2)$. A *strongly connected component (SCC)* of $\mathcal{A}$ is a maximal subset $C \subseteq Q$ in which every state is reachable from every state. Note that the term *component* will be used in a more general sense later.

Forward Powerset Complementation. The standard complementation first uses the *powerset construction* to transform a given NFA $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ into an equivalent complete DFA $det(\mathcal{A}) = (Q', \Sigma, \delta', I', F')$, where $Q' = 2^Q$, $\delta' = \{P \xrightarrow{a} \delta(P, a) \mid P \in Q', a \in \Sigma\}$, $I' = \{I\}$, and $F' = \{P \in Q' \mid P \cap F \neq \emptyset\}$. In the following, we assume that $det(\mathcal{A})$ does not contain unreachable states (only the reachable part of Q' is constructed). This does not improve the upper bound on the size of the DFA, which is still $2^{|Q|}$. Given a complete DFA $\mathcal{D} = (Q, \Sigma, \delta, I, F)$, its complement can be constructed as $co(\mathcal{D}) = (Q, \Sigma, \delta, I, Q \setminus F)$. A complement of an NFA $\mathcal{A}$ can thus be constructed as $co(det(\mathcal{A}))$. We call this construction *forward powerset complementation*.

3 Reverse Powerset Complementation

The idea of this approach to complementation is simple: if we reverse an automaton, then complement it (for example using the powerset complementation given above) and reverse again, we obtain the complement of the original automaton. Formally, given an NFA $\mathcal{A}$, its complement can be constructed as $rev(co(det(rev(\mathcal{A}))))$. We call this approach *reverse powerset complementation*. Fig. 1 shows all phases of this complementation process on an example automaton.

Contrary to the forward powerset construction, this method can yield a (forward-) nondeterministic automaton, which may be significantly smaller than the minimal deterministic complement. This can be documented by a generalization of the automata in Fig. 1. For any $n \in \mathbb{N}$, there exists an NFA $\mathcal{A}_n$ with $n+2$ states that accepts the language $\{a, b\}^*.\{a\}.\{a, b\}^n$. While the minimal forward powerset complement of $\mathcal{A}_n$ has 2^{n+1} states (intuitively, the DFA must store a bit vector of $n+1$ elements tracking which of

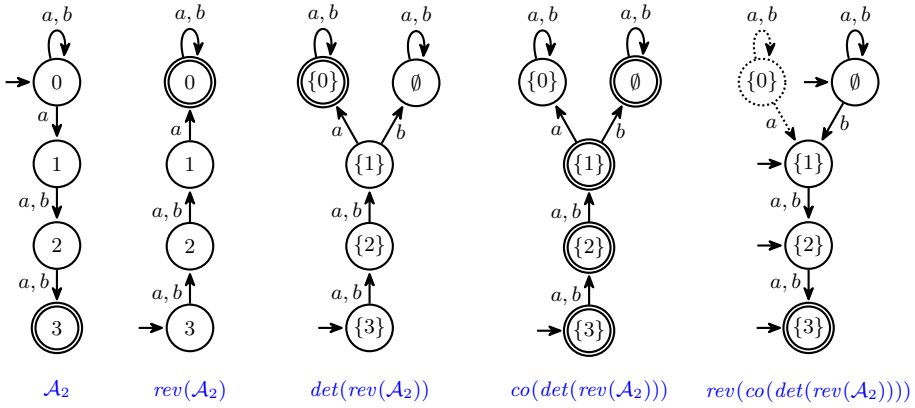

Fig. 1: NFA $\mathcal{A}_2$ accepting language $\{a,b\}^*.\{a\}.\{a,b\}^2$ and all phases of its reverse powerset complementation; the dotted part of the complement automaton is unreachable.

the last $n+1$ symbols read were a), the reverse powerset complementation produces an NFA with $n+2$ reachable states. This is due to $\mathcal{A}_n$ being reverse-deterministic. Our experiments in [21] show that the reverse powerset construction outperforms the forward one in many cases.

Heuristic for Forward vs. Reverse Powerset. Naturally, the powerset construction can be efficient in one direction (forward or reverse) while causing a blow-up in the other. To address this, we designed a cheap heuristic to choose the more favorable direction for a given NFA, for cases when running a portfolio is too expensive.

The heuristic sums the sizes of all powerset successors of each state in a given NFA. Formally, for an NFA $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, we define $sc(q) = \{\delta(q,a) \mid a \in \Sigma\}$ for each state $q \in Q$ and compute $powSC(\mathcal{A}) = |I| + \sum_{q \in Q} \sum_{S \in sc(q)} |S|$. We emphasize that if q has the same successors under two different symbols $a, b \in \Sigma$, then the set $\delta(q,a) = \delta(q,b)$ contributes to the sum only once.

Intuitively, a higher value of $powSC(\mathcal{A})$ should indicate a higher number of states produced by the powerset construction applied to $\mathcal{A}$. Hence, comparing $powSC(\mathcal{A})$ and $powSC(rev(\mathcal{A}))$ gives a hint which of the powerset constructions has a greater risk of blow-up. The heuristic's performance is experimentally evaluated in [21].

4 Sequential and Gate Complementation

This section introduces the basic ideas of two techniques called *sequential* and *gate complementation* with use on automata with a specific shape. The general form of these techniques is presented in Section 5.

Consider an NFA $\mathcal{A}$ that can be seen as two disjoint NFAs $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, I_1, \{q_F\})$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, \{q_I\}, F_2)$ connected with a single transition $q_F \xrightarrow{c} q_I$ for some $c \in \Sigma$, i.e., $\mathcal{A} = (Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2 \cup \{q_F \xrightarrow{c} q_I\}, I_1, F_2)$. The automaton $\mathcal{A}$ accepts the language $L_1.\{c\}.L_2$ where $L_1 = \mathcal{L}(\mathcal{A}_1)$ and $L_2 = \mathcal{L}(\mathcal{A}_2)$. An example of such an NFA $\mathcal{A}$ and the corresponding automata $\mathcal{A}_1$ and $\mathcal{A}_2$ can be found in Fig. 2. The transition $q_F \xrightarrow{c} q_I$ is called the *transfer transition* and automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are referred to as the *front* and the *rear component* of $\mathcal{A}$, respectively.

4.1 Sequential Complementation

Sequential complementation builds a potentially nondeterministic complement $\mathcal{C}$ of $\mathcal{A}$ from a complete DFA $\mathcal{A}'_1 = det(\mathcal{A}_1)$ equivalent to $\mathcal{A}_1$ and an NFA complement $\mathcal{C}_2$ of $\mathcal{A}_2$. The technique is called sequential complementation because we first complement $\mathcal{A}_2$ using an arbitrary complementation approach, and only then we build the complement of $\mathcal{A}$.

The construction is based on the following observation. The language $co(L_1.\{c\}.L_2)$ consists of words w such that, for all pairs of words u, v satisfying $ucv = w$, if $u \in L_1$ then $v \in co(L_2)$. Note that if w does not contain any c, the condition is trivially satisfied. Therefore, $\mathcal{C}$ will simulate $\mathcal{A}'_1$ and whenever a final state of $\mathcal{A}'_1$ is visited, that is, $\mathcal{C}$ finished reading a prefix $u \in L_1$, a transition under c initiates a new instance of $\mathcal{C}_2$ that will check that the corresponding suffix v is indeed in $co(L_2)$. Automaton $\mathcal{C}$ will accept if each initiated instance of $\mathcal{C}_2$ accepts, meaning that for all possible splittings of the input word w into ucv where $u \in L_1$, it holds that $v \in co(L_2)$.

Formally, let $\mathcal{A}'_1 = (Q'_1, \Sigma, \delta'_1, \{q'_0\}, F'_1)$ be a complete DFA representing the language L_1 and $\mathcal{C}_2 = (\widetilde{Q}_2, \Sigma, \widetilde{\delta}_2, \widetilde{I}_2, \widetilde{F}_2)$ be an NFA representing $co(L_2)$. We construct an NFA $\mathcal{C} = (\widetilde{Q}, \Sigma, \widetilde{\delta}, \widetilde{I}, \widetilde{F})$ representing $co(L_1.\{c\}.L_2)$ as follows.

- $\mathcal{C}$'s states are pairs composed of the current state of $\mathcal{A}'_1$ and states of instances of $\mathcal{C}_2$, $\widetilde{Q} = Q'_1 \times 2^{\widetilde{Q}_2}$.
- For each $a \in \Sigma$, the transition relation $\widetilde{\delta}$ simulates the corresponding transition of $\mathcal{A}'_1$ and arbitrary transitions under a of the running instances of $\mathcal{C}_2$. Moreover, it initiates a new instance of $\mathcal{C}_2$ whenever $\mathcal{A}'_1$ moves from an accepting state by reading c. Formally, for each $(p, \{r_1, \ldots, r_n\}) \in \widetilde{Q}$, $a \in \Sigma$, and transitions $p \xrightarrow{a} q \in \delta'_1$, $r_i \xrightarrow{a} s_i \in \widetilde{\delta}_2$ for all $1 \leq i \leq n$, the transition relation $\widetilde{\delta}$ contains transitions
  - $(p, \{r_1, \ldots, r_n\}) \xrightarrow{a} (q, \{s_1, \ldots, s_n\})$ if $p \notin F'_1$ or $a \neq c$, and
  - $(p, \{r_1, \ldots, r_n\}) \xrightarrow{a} (q, \{s_1, \ldots, s_n\} \cup \{s_0\})$ for all $s_0 \in \widetilde{I}_2$ if $p \in F'_1$ and $a = c$.
- $\mathcal{C}$ starts in the initial state q'_0 of $\mathcal{A}'_1$ with no running instance of $\mathcal{C}_2$, i.e., $\widetilde{I} = \{(q'_0, \emptyset)\}$.
- $\mathcal{C}$ accepts whenever all running instances of $\mathcal{C}_2$ accept, i.e., $\widetilde{F} = Q'_1 \times 2^{\widetilde{F}_2}$.

Theorem 1. *The NFA $\mathcal{C}$ accepts $co(\mathcal{L}(\mathcal{A}))$.*

Proof (sketch). Recall that $\mathcal{L}(\mathcal{A}) = L_1.\{c\}.L_2$. First, consider $w \in \mathcal{L}(\mathcal{A})$. Then there are $u \in L_1 = \mathcal{L}(\mathcal{A}'_1)$ and $v \in L_2$ such that $w = ucv$. As $\mathcal{A}'_1$ is deterministic, it has to reach a state $q_f \in F'_1$ after reading u. Hence, $\mathcal{C}$ can reach only states of the form (q_f, R) after reading u. When $\mathcal{C}$ reads c from this state, it reaches a state (q', R'), where R' has to contain some initial state s_0 of $\mathcal{C}_2$. However, $v \in L_2$ implies that $\mathcal{C}_2$ does not accept v. Hence, each state (q'', R'') of $\mathcal{C}$ reached from (q', R') by reading v is not accepting as it cannot satisfy $R'' \subseteq \widetilde{F}_2$. To sum up, $\mathcal{C}$ has no accepting run over $ucv = w$.

Now assume that $w \notin \mathcal{L}(\mathcal{A})$. As $\mathcal{A}'_1$ is deterministic and complete, it has a single run over w. Whenever the run reaches an accepting state over some prefix u of w, we know that $u \in L_1$ and thus the corresponding suffix cannot be of the form $cv \in \{c\}.L_2$ as that would contradict the assumption $w = ucv \notin \mathcal{L}(\mathcal{A})$. In other words, if the prefix u is followed by c, then $\mathcal{C}_2$ has an accepting run over the corresponding suffix v as $v \notin L_2$. We can construct an accepting run of $\mathcal{C}$ over w such that whenever the automaton $\mathcal{A}'_1$ tracked in the first element of the states of $\mathcal{C}$ reaches an accepting state and $\mathcal{C}$ reads c, we

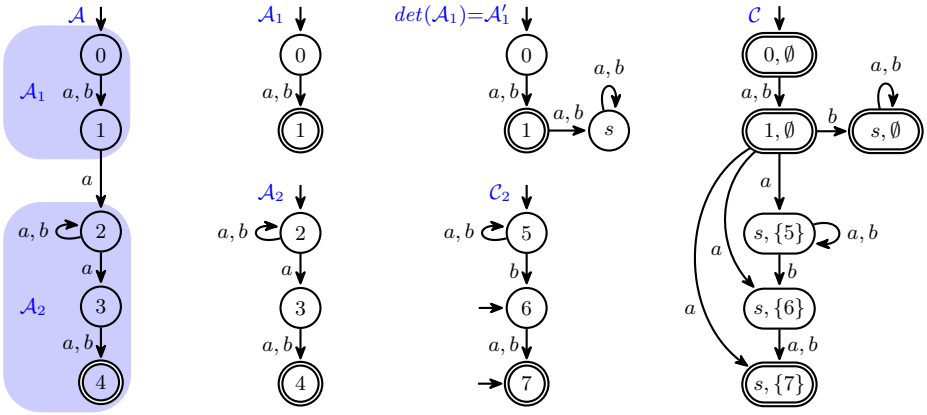

Fig. 2: An NFA $\mathcal{A}$, its front and rear components $\mathcal{A}_1, \mathcal{A}_2$, a complete DFA $\mathcal{A}'_1$ equivalent to $\mathcal{A}_1$, a complement $\mathcal{C}_2$ of $\mathcal{A}_2$, and the complement $\mathcal{C}$ of $\mathcal{A}$ constructed from $\mathcal{A}'_1$ and $\mathcal{C}_2$.

add to the second element of the state of $\mathcal{C}$ the initial state of the corresponding accepting run of $\mathcal{C}_2$ and then follow this run in the future transitions of $\mathcal{C}$. After reading the whole w, the second element of the reached state of $\mathcal{C}$ will contain only accepting states of $\widetilde{F}_2$. Thus, the constructed run of $\mathcal{C}$ over w is accepting and $w \in \mathcal{L}(\mathcal{C})$. □

Fig. 2 shows sequential complementation of an automaton $\mathcal{A}$ built from $\mathcal{A}_1$ and $\mathcal{A}_2$ connected by the transition $1 \xrightarrow{a} 2$ and illustrates that this complementation can produce nondeterministic results. Moreover, there exist automata for which the sequential complementation produces complements of linear size while both forward and reverse powerset complementations produce exponential complements. Consider the language $L_n = \{a,b\}^n.\{a\}.\{a,b\}^*.\{a\}.\{a,b\}^n$ for any $n \in \mathbb{N}$. There exists an automaton $\mathcal{B}_n$ with $2n+3$ states that accepts L_n (the automaton $\mathcal{B}_1$ is actually the automaton $\mathcal{A}$ in Fig. 2). Analogously to the figure, $\mathcal{B}_n$ can be decomposed into $\mathcal{B}_{n1}$ and $\mathcal{B}_{n2}$ accepting $L_{n1} = \{a,b\}^n$ and $L_{n2} = \{a,b\}^*.\{a\}.\{a,b\}^n$, respectively. Complementing $\mathcal{B}_{n2}$ into $\mathcal{C}_{n2}$ via the reverse powerset and applying sequential complementation yields a complement $\mathcal{C}_n$ with $2n+4$ states. In contrast, complementing $\mathcal{B}_n$ directly with either powerset method leads to an exponential blow-up – both results have $2^{n+1}+n+1$ states due to the loop under a, b and the nondeterminism in the middle of $\mathcal{B}_n$. Moreover, $\mathcal{B}_n$ belongs to an NFA class for which we prove a subexponential upper bound on the sequential complement size in [21]. This upper bound is strictly better compared to forward powerset for this NFA class.

4.2 Gate Complementation

Recall that we consider an automaton $\mathcal{A}$ that can be seen as two disjoint automata $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, I_1, \{q_F\})$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, \{q_I\}, F_2)$ connected by a transfer transition $q_F \xrightarrow{c} q_I$ for some $c \in \Sigma$, i.e., $\mathcal{A} = (Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2 \cup \{q_F \xrightarrow{c} q_I\}, I_1, F_2)$. Now we additionally assume that the symbol c of the transfer transition does not appear in any transition of the front component $\mathcal{A}_1$. The transfer transition is then called a *gate* and c is the *gate symbol*. A scheme and an example of an NFA $\mathcal{A}$ with a gate can be found in Fig. 3.

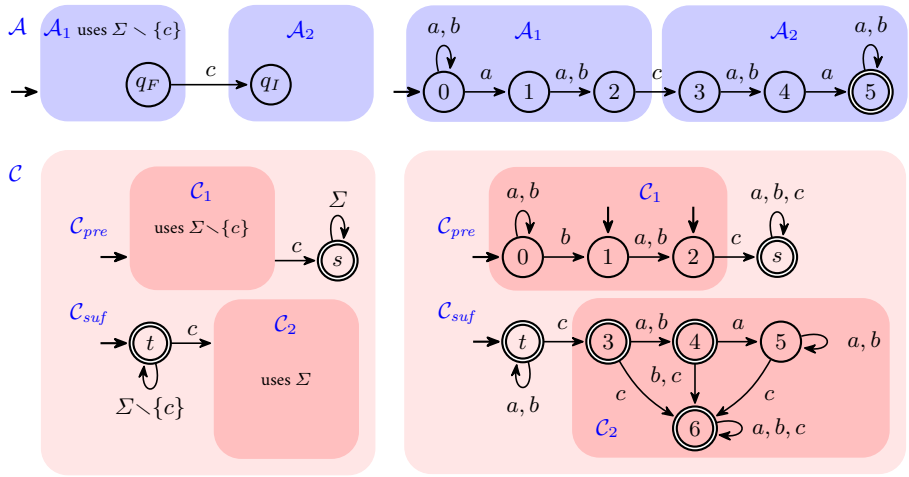

Fig. 3: A generic scheme of an NFA $\mathcal{A}$ with a gate (top left), a scheme of its complement $\mathcal{C}$ (bottom left), an instance of $\mathcal{A}$ for language $\{a,b\}^*.\{a\}.\{a,b\}.\{c\}.\{a,b\}.\{a\}.\{a,b\}^*$ (top right), and its gate complement $\mathcal{C}$ (bottom right).

Gate complementation utilizes the specific properties of $\mathcal{A}$. The automaton $\mathcal{A}$ accepts the language $L_1.\{c\}.L_2$ where $L_1 = \mathcal{L}(\mathcal{A}_1)$ and $L_2 = \mathcal{L}(\mathcal{A}_2)$. A word $w \in \Sigma^*$ belongs to $co(L_1.\{c\}.L_2)$ in the following cases:

1. w does not contain any c,
2. $w = ucv$ where $u \in (\Sigma \setminus \{c\})^*$ and $u \notin L_1$, or
3. $w = ucv$ where $u \in (\Sigma \setminus \{c\})^*$ and $v \notin L_2$.

Let $\mathcal{C}_1 = (\widetilde{Q}_1, \Sigma \setminus \{c\}, \widetilde{\delta}_1, \widetilde{I}_1, \widetilde{F}_1)$ be an arbitrary complement of $\mathcal{A}_1$ with respect to the alphabet $\Sigma \setminus \{c\}$ and $\mathcal{C}_2 = (\widetilde{Q}_2, \Sigma, \widetilde{\delta}_2, \widetilde{I}_2, \widetilde{F}_2)$ be an arbitrary complement of $\mathcal{A}_2$ with respect to the alphabet Σ. Then, the complement $\mathcal{C}$ of $\mathcal{A}$ consists of two parts: $\mathcal{C}_{pre}$ accepting the words according to Case 2 and $\mathcal{C}_{suf}$ accepting the words according to Cases 1 and 3. Schemes of $\mathcal{C}$, $\mathcal{C}_{pre}$, and $\mathcal{C}_{suf}$ can be seen in Fig. 3.

Formally, we set $\mathcal{C} = \mathcal{C}_{pre} \Cup \mathcal{C}_{suf}$ where $C_{pre} = (\widetilde{Q}_1 \cup \{s\}, \Sigma, \delta_{pre}, \widetilde{I}_1, \{s\})$ and $C_{suf} = (\widetilde{Q}_2 \cup \{t\}, \Sigma, \delta_{suf}, \{t\}, \{t\} \cup \widetilde{F}_2)$ such that s, t are fresh states,

$$\delta_{pre} = \widetilde{\delta}_1 \cup \{p \xrightarrow{c} s \mid p \in \widetilde{F}_1\} \cup \{s \xrightarrow{a} s \mid a \in \Sigma\}, \text{ and}$$
$$\delta_{suf} = \widetilde{\delta}_2 \cup \{t \xrightarrow{c} p \mid p \in \widetilde{I}_2\} \cup \{t \xrightarrow{a} t \mid a \in \Sigma \setminus \{c\}\}.$$

Theorem 2. *The NFA $\mathcal{C}$ accepts $co(\mathcal{L}(\mathcal{A}))$.*

Proof (sketch). Recall that $\mathcal{L}(A) = L_1.\{c\}.L_2$. First, consider $w \in \mathcal{L}(\mathcal{A})$. Then $w = ucv$, where $u \in L_1$ and $v \in L_2$. Because $u \in L_1$, no run of $\mathcal{C}_1$ over u reaches $\widetilde{F}_1$. Hence, no run of $\mathcal{C}_{pre}$ over a word starting with uc reaches s and thus $w \notin \mathcal{L}(\mathcal{C}_{pre})$. The run of $\mathcal{C}_{suf}$ over uc reaches $\widetilde{I}_2$ and it cannot be prolonged into an accepting run over w as $\mathcal{C}_2$ does not accept v. Altogether, we get $w \notin \mathcal{L}(\mathcal{C}_{pre}) \cup \mathcal{L}(\mathcal{C}_{suf}) = \mathcal{L}(\mathcal{C})$.

Now assume that $w \notin \mathcal{L}(\mathcal{A})$. If w does not contain any c, it is accepted by $\mathcal{C}_{suf}$. Let $w = ucv$, where $u \in (\Sigma \setminus \{c\})^*$. As $w \notin \mathcal{L}(\mathcal{A})$, we know that $u \notin L_1$ or $v \notin L_2$. In the former case, $u \in \mathcal{L}(\mathcal{C}_1)$ and thus w is accepted by $\mathcal{C}_{pre}$. In the latter case, $v \in \mathcal{L}(\mathcal{C}_2)$ and thus w is accepted by $\mathcal{C}_{suf}$. To sum up, $w \in \mathcal{L}(\mathcal{C}_{pre}) \cup \mathcal{L}(\mathcal{C}_{suf}) = \mathcal{L}(\mathcal{C})$. □

Note that $\mathcal{C}$ has only $|\mathcal{C}_1| + |\mathcal{C}_2| + 2$ states. To see the advantage of this complementation approach, consider $L_n = L_{n1}.\{c\}.L_{n2}$, where $L_{n1} = \{a,b\}^*.\{a\}.\{a,b\}^n$, $L_{n2} = \{a,b\}^n.\{a\}.\{a,b\}^*$, and $n \in \mathbb{N}$. There exists an NFA $\mathcal{B}_n$ with only $2n+4$ states accepting L_n, which can be deconstructed into two components accepting L_{n1} and L_{n2}, respectively (the automaton $\mathcal{B}_1$ is actually the automaton $\mathcal{A}$ in Fig. 3). If we use reverse powerset to complement the front component, forward powerset for the rear one, and combine the outputs by gate complementation, the result $\mathcal{C}_n$ has $2n+7$ states. Complementing the whole NFA $\mathcal{B}_n$ with either forward or reverse powerset causes an exponential blow-up, with both results having $2^{n+1}+n+2$ states. Sequential complementation (Section 4.1) also results in a blow-up (for all possible divisions of $\mathcal{B}_n$ into front and rear components) due to the determinization of the front component and/or tracking possibly many instances of the complement of the rear component.

5 Generalized Complementation Problem

This section briefly generalizes the ideas from Section 4. The generalized complementation constructions are still applicable to automata consisting of two components, but there can be an arbitrary number of transfer transitions under various symbols leading from the front to the rear component. The generalized constructions again use complements of the components. These complements can be constructed either by forward or reverse powerset complementation, or by a recursive application of (generalized) sequential or gate complementation.

Due to the potentially recursive application of our complementation constructions and due to the fact that components can be connected by multiple transfer transitions, we need to complement components with many incoming and outgoing transfer transitions. Therefore, we work with automata that generalize initial and final states to multiple sets of entry and exit states. These sets are called *entry* and *exit port sets*. We talk about *port automata* and generalize the complementation problem to these automata as follows.

A *port NFA* or simply a *port automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \mathcal{I}, \mathcal{F})$, where Q, Σ, and δ are as in an NFA, and $\mathcal{I} = (I_0, \ldots, I_k)$ and $\mathcal{F} = (F_0, \ldots, F_\ell)$ are sequences of subsets of Q called *entry port sets*, resp. *exit port sets*. A *slice* of $\mathcal{A}$ is an NFA with one entry and one exit port set chosen as the initial, resp. final states, i.e., the NFA $\mathcal{A}^{i,j} = (Q, \Sigma, \delta, I_i, F_j)$ for $0 \leq i \leq k$ and $0 \leq j \leq \ell$. $\mathcal{A}$ is *deterministic* (*port DFA*) if all its slices are deterministic (in particular, $|I_i| = 1$ must hold for every $0 \leq i \leq k$). A port DFA is *complete* if all its slices are complete. A *complement* of $\mathcal{A}$ is a port NFA representing complements of all slices. More precisely, a complement of $\mathcal{A}$ is a port NFA $\mathcal{C} = (\widetilde{Q}, \Sigma, \widetilde{\delta}, \widetilde{\mathcal{I}}, \widetilde{\mathcal{F}})$ with $\widetilde{\mathcal{I}} = (\widetilde{I}_0, \ldots, \widetilde{I}_k)$ and $\widetilde{\mathcal{F}} = (\widetilde{F}_1, \ldots, \widetilde{F}_\ell)$ such that $\mathcal{L}\left(\mathcal{C}^{i,j}\right) = co(\mathcal{L}\left(\mathcal{A}^{i,j}\right))$ for each $0 \leq i \leq k$ and $0 \leq j \leq \ell$. We call $\widetilde{I}_i$ an *entry port complement* of I_i and $\widetilde{F}_j$ an *exit port complement* of F_j, together shortened to *port complements*.

In the rest of this section, we first generalize the powerset construction to port automata to get a determinization procedure needed in the sequential complementation.

With that, we generalize the forward and reverse powerset complementation to port automata. Powerset complementations are applied to components that cannot be recursively complemented by sequential or gate complementation (for example, because they cannot be further decomposed into a front and a rear component). Finally, we outline general versions of sequential and gate complementations.

5.1 Powerset Construction and Complementation for Port Automata

The powerset construction and complementation generalize to port automata as follows. Given a port NFA $\mathcal{A} = (Q, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ with $\mathcal{I} = (I_i)_{0 \leq i \leq k}$ and $\mathcal{F} = (F_j)_{0 \leq j \leq \ell}$, the *port powerset construction* produces a port DFA $det(\mathcal{A}) = (Q', \Sigma, \delta', \mathcal{I}', \mathcal{F}')$, where Q' and δ' are defined as in the standard powerset construction (see Section 2), $\mathcal{I}' = (\{I_i\})_{0 \leq i \leq k}$, and $\mathcal{F}' = (\{P \in Q' \mid P \cap F_j \neq \emptyset\})_{0 \leq j \leq \ell}$. The original $\mathcal{A}$ and $det(\mathcal{A})$ are equivalent, i.e., $\mathcal{L}\left(\mathcal{A}^{i,j}\right) = \mathcal{L}\left(det(\mathcal{A})^{i,j}\right)$ for all $0 \leq i \leq k$ and $0 \leq j \leq \ell$. Moreover, $det(\mathcal{A})$ is *complete*. The *complement* of any complete port DFA $\mathcal{D} = (Q', \Sigma, \delta', \mathcal{I}', \mathcal{F}')$ with $\mathcal{F}' = (F'_j)_{0 \leq j \leq \ell}$ is the port DFA $co(\mathcal{D}) = (Q', \Sigma, \delta', \mathcal{I}', \mathcal{F}'')$ where $\mathcal{F}'' = (Q' \setminus F'_j)_{0 \leq j \leq \ell}$. The forward powerset complement of a port NFA is, as for non-port NFAs, defined as $co(det(\mathcal{A}))$. The *reverse* of a port NFA $\mathcal{A}$ is the port NFA $rev(\mathcal{A}) = (Q, \Sigma, rev(\delta), \mathcal{F}, \mathcal{I})$. The reverse powerset complement is then constructed as $rev(co(det(rev(\mathcal{A}))))$.

5.2 Generalized Sequential Complementation

We now outline the generalization of sequential complementation to port NFAs, highlighting only the differences from Section 4.1 (see [21] for details). Let $\mathcal{A}$ be a component constructed by merging $\mathcal{A}_1$ and $\mathcal{A}_2$. Hence, $\mathcal{A}$ is a port NFA consisting of the front component $\mathcal{A}_1$ and the rear component $\mathcal{A}_2$, both port NFAs, connected by a set of transfer transitions δ_{trans} leading from $\mathcal{A}_1$ to $\mathcal{A}_2$. Unlike Section 4.1, we have a set of transfer transitions δ_{trans} instead of a single one, and both components have multiple entry/exit port sets. We first construct $det(\mathcal{A}_1)$ and a complement port NFA $\mathcal{C}_2$ of $\mathcal{A}_2$. Analogously to Section 4.1, the constructed complement $\mathcal{C}$ of $\mathcal{A}$ contains states (q, R), where q tracks the only run of $det(\mathcal{A}_1)$ and R tracks runs of $\mathcal{C}_2$ over suffixes of the input word.

The generalized construction closely follows the basic one. It differs in the following:

1. Given a state q of $det(\mathcal{A}_1)$ and a symbol $a \in \Sigma$, let P be the set of states p of $\mathcal{A}_2$ such that $\mathcal{A}$ contains a transfer transition $q'' \xrightarrow{a} p$ from some $q'' \in q$. Whenever $\mathcal{C}$ reaches a state (q, R) with a next on input, it has to check that the rest of the word is not accepted by $\mathcal{A}_2$ starting from any $p \in P$ and thus it spawns a new instance of $\mathcal{C}_2$ for each $p \in P$. Formally, $\mathcal{C}$ has all transitions $(q, R) \xrightarrow{a} (q', R')$ where (1) $q \xrightarrow{a} q'$ is the transition from q under a in $det(\mathcal{A}_1)$, (2) for each $r \in R$, R' contains some s such that $r \xrightarrow{a} s$ is a transition in $\mathcal{C}_2$, and, additionally, (3) for every $p \in P$, R' contains some state s from the port complement of the newly added entry port set $\{p\}$.
2. Since $\mathcal{A}_1$ may have exit ports with transitions leaving $\mathcal{A}$ and $\mathcal{A}_2$ may have entry ports with incoming edges from outside, $\mathcal{C}$ must reject words accepted entirely within either $\mathcal{A}_1$ or $\mathcal{A}_2$. Therefore, if a slice $\mathcal{A}^{i,j}$ has entry ports in $\mathcal{A}_2$, we activate one instance of $\mathcal{C}_2^{i,j}$ at the start: $\mathcal{C}^{i,j}$ has entry ports of the form $(q, \{r\})$, where q is the

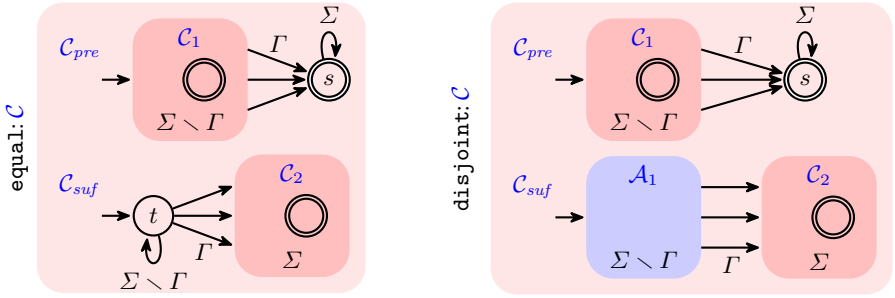

Fig. 4: Gate complement using the `equal` (left) and `disjoint` method (right).

only entry port in $det(\mathcal{A}_1)^{i,j}$ and r is an entry port in $\mathcal{C}_2^{i,j}$. At the same time, a state (q, R) is an exit port of $\mathcal{C}^{i,j}$ only if q is not an exit port of $det(\mathcal{A}_1)^{i,j}$, ensuring the input word was not accepted in $\mathcal{A}_1^{i,j}$.

5.3 Generalized Gate Complementation

Here we outline the core ideas of generalized gate complementation as an extension of Section 4.2, with full details in [21]. Recall that in Section 4.2, the NFA $\mathcal{A}$ was assumed to consist of two components connected by a single gate transition labeled with a gate symbol not occurring in the front component. We now generalize this by allowing arbitrary transitions leading from the front to the rear component and labeled with gate symbols $\Gamma \subseteq \Sigma$ such that $\mathcal{A}_1$ has no transitions under Γ. We call a gate transition labeled with c a c-gate. We also suppose that the input NFA may be a port NFA. We present two variants of the algorithm, `equal` and `disjoint` (illustrated in Fig. 4), each adding further constraints on the input NFA.

The simpler variant `equal` more closely resembles Section 4.2. If we write the words read by $\mathcal{A}$ as ucv with $u \in (\Sigma \setminus \Gamma)^*$, $c \in \Gamma$, and $v \in \Sigma^*$, this variant assumes that for every gate symbol $c \in \Gamma$, every two entry ports of $\mathcal{A}_2$ with an incoming c-gate can be reached by reading exactly the same prefixes of the form uc. To recognize all prefixes that cannot be read in $\mathcal{A}_1$ and followed by c, we collect the ports of $\mathcal{A}_1$ with outgoing c-gates into a new exit port set. We do the same for the suffixes that are not accepted in $\mathcal{A}_2$ from states with an incoming c-gate, and give $\mathcal{A}_2$ a new entry port set consisting of those states. These new port sets (one for each gate symbol) are preserved through complementation, and in $\mathcal{C}_{pre}$ and $\mathcal{C}_{suf}$, we can connect their complement port sets in $\mathcal{C}_1$ and $\mathcal{C}_2$ to the states s and t using an appropriate gate symbol, respectively. This way, $\mathcal{C}_{pre}$ accepts words where the first-appearing gate symbol c follows an invalid prefix and $\mathcal{C}_{suf}$ accepts words where c is followed by an invalid suffix.

The `disjoint` variant relaxes the requirement that all $\mathcal{A}_2$'s entry ports with an incoming c-gate must be reached by the same words of the form uc. Instead, $\mathcal{C}_{suf}$ tracks used gates, assuming the complemented port NFA is partitioned as follows: for any $c \in \Gamma$ and any two gates $p \xrightarrow{c} r$, $p' \xrightarrow{c} r'$, if the languages accepted by p and p' in any slice $\mathcal{A}_1^{i,j}$ are not disjoint, then the languages accepted from r and r' in the slice $\mathcal{A}_2^{i,j}$ must be equal. Unlike the `equal` method, $\mathcal{C}_{suf}$ now depends on both $\mathcal{A}_1$ and $\mathcal{C}_2$ to track the used gate.

6 Implementation and Evaluation

We have implemented the described algorithms in a tool called `AliGater`[4] written in Python and using the C++ library `Mata` [9] and the Python library `Automata` [12] as backends. `AliGater` also integrates the `Reduce` tool [26] for NFA reduction.

`AliGater` calls `Mata` for forward (`fwd powerset`) and reverse (`rev powerset`) powerset complementation, and Hopcroft's minimization [22,30] (denoted by the suffix `+ min`). Sequential (`seq`) and gate (`gate`) complementations are implemented in `AliGater` in their generalized versions, described in Sections 5.2 and 5.3. Selected implementation details for both of these methods are discussed below. Other details (e.g., the use of reductions) and an extended description of `AliGater` settings used in the evaluation are available in [21].

6.1 Implementation of Sequential Complementation

The implementation of sequential complementation first divides the NFA $\mathcal{A}$ into components $\mathcal{A}_1, \ldots, \mathcal{A}_n$. Then $\mathcal{A}_n$ is complemented using forward or reverse powerset complementation, and $\mathcal{A}_1, \ldots, \mathcal{A}_{n-1}$ are determinized. The automaton $det(\mathcal{A}_{n-1})$ is then composed with the complement of $\mathcal{A}_n$ to form a new complemented bottom component; the same process is repeated with every preceding component up to $\mathcal{A}_1$.

We implemented three approaches to divide $\mathcal{A}$ into components:

1. *Deterministic components*. Because the components $\mathcal{A}_1, \ldots, \mathcal{A}_{n-1}$ are determinized during sequential complementation, this partitioning approach tries to avoid determinization and use transfer transitions to cover some nondeterminism in $\mathcal{A}$. We first decompose $\mathcal{A}$ into SCCs and order them by topological ordering. If the first SCC is nondeterministic, we set $\mathcal{A}_1$ to be this SCC. If it is deterministic, we set $\mathcal{A}_1$ to be the first SCC and we repeatedly add previously unused SCCs in the order of the topological ordering (with the corresponding transfer transitions) until a maximal deterministic $\mathcal{A}_1$ is obtained. Note that $\mathcal{A}_1$ may not be connected. The rest of the automaton $\mathcal{A}$ is then recursively divided in the same manner.
2. *Deterministic components + reverse-deterministic bottom component*. Sequential complementation requires $\mathcal{A}_n$ to be complemented, and if it is reverse-deterministic, we can complement it easily by reverse powerset complementation. We compute $\mathcal{A}_n$ as a maximal bottom reverse-deterministic part of $\mathcal{A}$, analogously to the computation of $\mathcal{A}_1$ in the previous approach. The rest of $\mathcal{A}$ is divided in the same way as above.
3. *Min-cut*. $\mathcal{A}$ is divided into two components, where the partition with the fewest transfer transitions from $\mathcal{A}_1$ to $\mathcal{A}_2$ is chosen. To obtain this partition, we construct a directed graph whose vertices are the SCCs of $\mathcal{A}$. The capacity of each edge is the amount of transitions between the SCCs. The function `minimum_cut()` from the `NetworkX` library [18] is then used to compute the minimum cut of this graph. The source is a fresh vertex with edges to all SCCs with no predecessors, the sink is the last SCC in a topological ordering.

[4] `https://gitlab.fi.muni.cz/xstepkov/aligater`

6.2 Implementation of Gate Complementation

`AliGater` implements the generalized gate complementation methods. The implementation tries to find all possible partitions of the input NFA $\mathcal{A}$ satisfying the input conditions of any of the methods presented in Section 5.3 (cf. [21] for more details). The input conditions are formulated as the language equivalence or disjointness of certain states and evaluated by `Mata`; language equivalence is tested using the *antichains algorithm* [34], language disjointness is checked via an emptiness check on the product automaton. The concrete details can be found in [21], but the main idea is to prefer partitions tagged with `equal` to those tagged with `disjoint`, since they usually deliver smaller results, and that we try to pick a partition where the two components have a similar size.

6.3 Evaluation

All experiments were run on computers with the Intel® Core™ i7-8700 CPU. Each complementation algorithm was executed on each input NFA with the timeout of 5 min and the memory limit of 8 GiB. We focus on the size (number of states) of the results and also how often the individual methods were successful (finished within the time and memory limits). TO and MO means that the time or memory limit was reached, respectively.

For the evaluation, we used a total of 9,450 benchmarks from `nfa-bench` [32], which gathers automata benchmarks from diverse applications. Additional details regarding the families of benchmarks are in [21].

Results. We evaluated the performance of the proposed algorithms `seq` and `gate` compared to the best result of `fwd powerset` + `min` and `rev powerset` + `min` (Fig. 5, left). The methods `seq` and `gate` are, in general, computationally more intensive than the powerset constructions. Running `seq` often produces large automata unless reduced during the process. The automata structure, however, allows `Reduce` to effectively reduce them, unlike automata generated by powerset constructions. Applicability of `gate` is limited by input conditions (it produced results for 2,577 benchmarks) and evaluating these conditions can be costly. Despite frequent timeouts, these methods sometimes achieve significantly better results than powerset-based methods, suggesting further room for improvement in NFA complementation beyond powerset-based techniques.

We have also evaluated the benefit of using all available techniques (e.g., in a portfolio) against `fwd powerset` + `min` as the baseline method (Fig. 5, right). This use case is targeted at applications where it pays off to obtain as small automaton for the complement as possible, such as when translating an extended regex into an NFA that will be used millions of times during matching. The results show that the proposed techniques were in many cases able to bring significant benefits, in particular solving a number of cases when `fwd powerset` + `min` ran out of resources.

7 Conclusion

We have presented, to the best of our knowledge, the first systematic empirical study of NFA complementation approaches. We suggested several novel algorithms for complementation of (subclasses of) NFAs. We carried out an extensive experimental evaluation of the approaches and showed that alternative complementation algorithms can often give a significantly better result than the classic approach (sometimes even in orders of

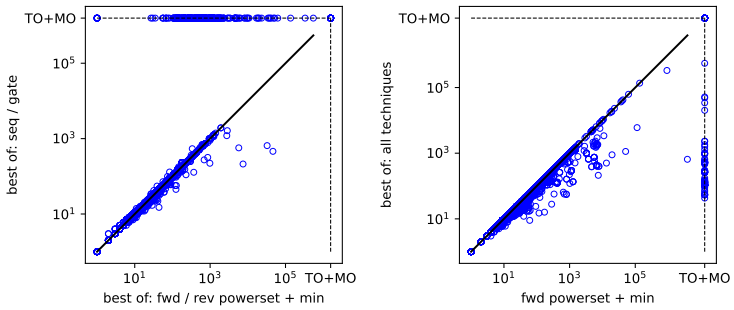

Fig. 5: Comparison of sizes of complements (number of states) given by the best of `fwd powerset + min` and `rev powerset + min` against the best of `seq` and `gate` (left), and `fwd powerset + min` against the best result of all techniques (right).

magnitude). We have also suggested a heuristic that helps to select between the classic approach and reverse powerset complementation.

There are still multiple opportunities for improvement, e.g., in the partitioning process for the sequential complementation. Moreover, other structural classes of automata amenable for efficient complementation may exist.

Acknowledgements

We thank the anonymous reviewers for careful reading of the paper and their suggestions that improved its quality. L. Holík and O. Lengál were supported by the Czech Ministry of Education, Youth and Sports ERC.CZ project LL1908, the Czech Science Foundation projects 23-07565S and 25-17934S, and the FIT BUT internal project FIT-S-23-8151. A. Štěpková and J. Strejček were supported by the European Union's Horizon Europe program under the grant agreement No. 101087529 (CHESS).

References

1. Allred, J.D., Ultes-Nitsche, U.: A simple and optimal complementation algorithm for Büchi automata. In: LICS'18. pp. 46–55. ACM (2018)
2. Ayari, A., Basin, D.A.: Bounded model construction for monadic second-order logics. In: CAV'00. LNCS, vol. 1855, pp. 99–112. Springer (2000)
3. Birget, J.: Partial orders on words, minimal elements of regular languages and state complexity. Theor. Comput. Sci. **119**(2), 267–291 (1993)
4. Blahoudek, F., Duret-Lutz, A., Strejček, J.: Seminator 2 can complement generalized Büchi automata via improved semi-determinization. In: CAV'20. Springer (2020)
5. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract regular (tree) model checking. Int. J. Softw. Tools Technol. Transf. **14**(2), 167–191 (2012)
6. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: CAV'00. LNCS, vol. 1855, pp. 403–418. Springer (2000)
7. Büchi, J.R.: On a decision method in restricted second-order arithmetic. In: Intl. Congress on Logic, Methodology, and Philosophy of Science. pp. 1–11. Stanford Univ. Press (1962)
8. Bustan, D., Grumberg, O.: Simulation-based Minimization. ACM Transactions on Computational Logic **4**(2), 181–206 (2003)

9. Chocholatý, D., Fiedor, T., Havlena, V., Holík, L., Hruška, M., Lengál, O., Síč, J.: Mata: A fast and simple finite automata library. In: TACAS'24. pp. 130–151. Springer, Cham (2024)
10. Chocholatý, D., Havlena, V., Holík, L., Hranička, J., Lengál, O., Síč, J.: Z3-Noodler 1.3: Shepherding decision procedures for strings with model generation. In: TACAS'25. Springer (2025)
11. Elgaard, J., Klarlund, N., Møller, A.: MONA 1.x: New techniques for WS1S and WS2S. In: CAV'98. LNCS, vol. 1427, pp. 516–520. Springer (1998)
12. Evans, C.: CALEB531/automata: A Python library for simulating finite automata, pushdown automata, and Turing machines (2025)
13. Fiedor, T., Holík, L., Janků, P., Lengál, O., Vojnar, T.: Lazy automata techniques for WS1S. In: TACAS'17. LNCS, vol. 10205, pp. 407–425 (2017)
14. Fiedor, T., Holík, L., Lengál, O., et al.: Nested antichains for WS1S. Acta Inform. **56**(3) (2019)
15. van Glabbeek, R.J., Ploeger, B.: Five determinisation algorithms. In: CIAA'08. Springer (2008)
16. Glenn, J., Gasarch, W.I.: Implementing WS1S via finite automata. In: WIA'96. Springer (1996)
17. Habermehl, P., Havlena, V., Hečko, M., Holík, L., Lengál, O.: Algebraic reasoning meets automata in solving linear integer arithmetic. In: CAV'24. pp. 42–67. Springer (2024)
18. Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using networkx. In: SciPy'08. pp. 11 – 15. Pasadena, CA USA (2008)
19. Havlena, V., Lengál, O., Šmahlíková, B.: Complementation of Emerson-Lei automata. In: FoSSaCS'25. LNCS, Springer (2025)
20. Holzer, M., Kutrib, M.: State complexity of basic operations on nondeterministic finite automata. In: CIAA'02. LNCS, vol. 2608, pp. 148–157. Springer (2002)
21. Holík, L., Lengál, O., Major, J., Štěpková, A., Strejček, J.: On complementation of nondeterministic finite automata without full determinization (technical report). CoRR **abs/2507.03439** (2025), `https://arxiv.org/abs/2507.03439`
22. Hopcroft, J.: An $n \log n$ algorithm for minimizing states in a finite automaton. In: Kohavi, Z., Paz, A. (eds.) Theory of Machines and Computations, pp. 189–196. Academic Press (1971)
23. Jirásková, G.: State complexity of some operations on binary regular languages. Theor. Comput. Sci. **330**(2), 287–298 (2005)
24. Kelb, P., Margaria, T., Mendler, M., Gsottberger, C.: MOSEL: A sound and efficient tool for M2L(Str). In: CAV'97. LNCS, vol. 1254, pp. 448–451. Springer (1997)
25. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E.: Symbolic model checking with rich assertional languages. Theor. Comput. Sci. **256**(1-2), 93–112 (2001)
26. Mayr, R., Clemente, L.: Advanced automata minimization. In: POPL'13. ACM (2013)
27. Meyer, A.R.: Weak monadic second order theory of successor is not elementary-recursive. In: Logic Colloquium. Lecture Notes in Mathematics, vol. 453, pp. 132–154. Springer (1972)
28. Rabin, M.O., Scott, D.S.: Finite automata and their dec. problems. IBM J. Res. Dev. **3**(2) (1959)
29. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two way finite automata. In: STOC'78. pp. 275–286. ACM (1978)
30. Valmari, A., Lehtinen, P.: Efficient minimization of DFAs with partial transition functions. In: STACS'08. LIPIcs, vol. 1, pp. 645–656. Dagstuhl (2008)
31. Varatalu, I.E., Veanes, M., Ernits, J.P.: RE#: High performance derivative-based regex matching with intersection, complement, and restricted lookarounds. PACMPL **9**(POPL), 1–32 (2025)
32. VeriFIT: `nfa-bench`: Extensive benchmark for reasoning about regular properties (2025), `https://github.com/VeriFIT/nfa-bench`
33. Wolper, P., Boigelot, B.: An automata-theoretic approach to Presburger arithmetic constraints (extended abstract). In: SAS'95. LNCS, vol. 983, pp. 21–32. Springer (1995)
34. Wulf, M.D., Doyen, L., Henzinger, T.A., Raskin, J.: Antichains: A new algorithm for checking universality of finite automata. In: CAV'06. LNCS, vol. 4144, pp. 17–30. Springer (2006)