When Does Abstraction Help?

Guy Avni and Orna Kupferman

School of Computer Science and Engineering, Hebrew University, Israel

Abstract

Abstraction is a leading technique for coping with large state spaces. Abstraction over-approximates the transitions of the original system or the automaton that models it and may introduce nondeterminism. In applications where determinism is essential, we say that an abstraction function is helpful if, after determining and minimizing the abstract automaton, we end up with fewer states than the original automaton. We show that abstraction functions are not always helpful; in fact, they may introduce an exponential blow-up. We study the problem of deciding whether a given abstraction function is helpful for a given deterministic automaton and show that it is PSPACE-complete.

Keywords: Formal methods, Abstraction, Deterministic finite automata

1. Introduction

The automata-theoretic approach has proven to be a very versatile and fruitful approach for formal reasoning about systems and their on-going behaviors. Automata are used in order to model both systems and specifications. One of the big challenges in practice is the need to reason about automata with huge state spaces. In *abstraction*, we cope with the huge state spaces by translating the automata to ones with smaller state spaces [1]. Typically, by hiding some of the information associated with each state, different states of the original automaton are mapped to the same abstract state.

Technically, if the original automaton $\mathcal{A}$ has state space Q, then the abstraction consists of a function $\alpha : Q \to A$, where A is the set of abstract states and is smaller than Q. The transitions in such abstractions are defined so that there is a transition with the letter σ from one abstract state a to the other abstract state a' if some concrete state that is mapped to a has a σ-transition to some concrete state that is mapped to a'. Such over-approximating abstractions are very useful in practice. In particular, if the language of an automaton whose language over-approximates the language of a system is contained in the language of the specification, then we can

conclude that the system satisfies the specification. Moreover, when the answer is negative, it is possible to refine the abstract automaton until a definite answer to the verification problem is obtained (see work on counterexample-guided abstraction refinement [2]).

In addition to extending the language of the original automaton, abstraction also increases its nondeterminism. In particular, it may be that the original automaton is deterministic (that is, each of its states has a single outgoing σ-transition for each letter σ) whereas the abstraction is nondeterministic. Indeed, if several concrete states, each with a different σ-successor, are mapped to the same abstract state a, then a may have several σ-successors.

The fact that abstraction does not preserve determinism is a serious drawback, as algorithms for deterministic automata are typically much simpler than ones in the nondeterministic setting. Also, for some problems, such as synthesis or reasoning about probabilistic systems, solutions are known only for deterministic automata, and determination is required when the input to the problem is nondeterministic [3, 4]. For some algorithms, such as trigger querying or reasoning about memoryful formalisms, determinism is essential not only in the specification but also in the system [5, 6].

In this work we ask whether, given the need to determinize the abstract automaton, abstraction still leads to smaller automata. Formally, consider a deterministic finite automaton (DFA, for short) $\mathcal{A}$, and let $\mathcal{A}_\alpha$ be a nondeterministic finite automaton (NFA, for short) obtained from $\mathcal{A}$ by applying an abstraction function α. Let $\mathcal{D}_\alpha$ be the minimal DFA equivalent to $\mathcal{A}_\alpha$. We ask whether $\mathcal{D}_\alpha$ is smaller than $\mathcal{A}$. If so, we say that α is helpful.

We show that, surprisingly, abstractions are not always helpful. In fact, we show a family of DFAs and abstraction functions for them for which the abstract automata are exponentially bigger than the original automata. We also study the problem of deciding whether a given abstraction function is helpful for a given DFA and show that it is PSPACE-complete.

2. Preliminaries

Automata. A nondeterministic finite automaton (NFA, for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$, where Σ is an alphabet, Q is a set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states. We refer to $|Q|$ as the *size* of $|\mathcal{A}|$, and we assume that it is complete: every state has an outgoing transition. We say that $\mathcal{A}$ is *deterministic* if $|Q_0| = 1$ and for every $q \in Q$ and $\sigma \in \Sigma$, we have $|\delta(q, \sigma)| \leq 1$. For $q, q' \in Q$ and $\sigma \in \Sigma$ with $q' \in \delta(q, \sigma)$, we say that q' is a σ-successor of q.

A run of $\mathcal{A}$ on a word $u = u_1, \ldots, u_n \in \Sigma^*$ is a sequence of states $r = r_0, r_1, \ldots, r_n$ such that $r_0 \in Q_0$ and for every $0 \leq i < n$ we have $r_{i+1} \in \delta(r_i, u_{i+1})$. The run r is accepting iff $r_n \in F$. Since $\mathcal{A}$ is non-deterministic, there can be more than one run on a single word. A word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on w. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words in Σ^* that $\mathcal{A}$ accepts.

Abstracting Automata. An *abstraction function* for an NFA $\mathcal{A}$ is a function $\alpha : Q \to A$, for a set A, which we assume to be smaller than Q. We refer to Q as the *concrete states* and to A as the set of *abstract states*. The function α induces a partition of Q, where two states q and q' are in the same set if $\alpha(q) = \alpha(q')$. We sometimes refer to abstract states as sets of concrete states. In particular, for a concrete state $c \in Q$ and an abstract state $a \in A$, we use the notation $c \in a$ to indicate that $\alpha(c) = a$.

Consider an NFA $\mathcal{A}$ and an abstraction function α. The abstraction of $\mathcal{A}$ according to α is the NFA $\mathcal{A}[\alpha] = \langle \Sigma, A, \delta_\alpha, A_0, F_\alpha \rangle$, where $A_0 = \{\alpha(q_0) : q_0 \in Q_0\}$, $F_\alpha = \{a \in A : a \cap F \neq \emptyset\}$, and δ_α is defined as follows: Consider abstract states $a, a' \in A$ and a letter $\sigma \in \Sigma$. We define $\delta_\alpha : A \times \Sigma \to 2^A$ so that $a' \in \delta_\alpha(a, \sigma)$ iff there exists $c \in a$ and $c' \in a'$ such that $c' \in \delta(c, \sigma)$. Note that $\mathcal{A}[\alpha]$ *over-approximates* $\mathcal{A}$ in the sense that each accepting run $r_0, r_1, \ldots, r_n$ of $\mathcal{A}$ on a word w induces an accepting run $\alpha(r_0), \alpha(r_1), \ldots, \alpha(r_n)$ of $\mathcal{A}[\alpha]$ on w. Thus, $L(\mathcal{A}) \subseteq L(\mathcal{A}[\alpha])$.

Abstraction Helps. Consider a DFA $\mathcal{A}$ and an abstraction function for it α. It is easy to see that even though $\mathcal{A}$ is deterministic, the automaton $\mathcal{A}[\alpha]$ may be nondeterministic. For example, if c and c' are states of $\mathcal{A}$ and α is an abstraction function with $\alpha(c) = \alpha(c')$ yet $\alpha(\delta(c, \sigma)) \neq \alpha(\delta(c', \sigma))$, then the abstract state $\alpha(c)$ has at least two σ-successors. Since DFAs are as expressive as NFAs, there is a DFA $\mathcal{D}$ such that $L(\mathcal{D}) = L(\mathcal{A}[\alpha])$. However, $\mathcal{D}$ may be exponentially bigger than $\mathcal{A}[\alpha]$. Moreover, even the minimal DFA that recognizes $L(\mathcal{A}[\alpha])$ may be big. We say that an abstraction is *helpful* if the size of the minimal DFA $\mathcal{D}'$ with $L(\mathcal{D}') = L(\mathcal{A}[\alpha])$ is at most the size of the original automaton $\mathcal{A}$.

Remark 1. Recall that $\mathcal{A}[\alpha]$ over-approximates $\mathcal{A}$. Researchers have also studied abstractions that under-approximate the original automaton. This type of abstraction preserves determinism and is thus always helpful in the sense studied here.

3. Abstraction is Not Always Helpful

Consider an NFA $\mathcal{A}$ and an abstraction function α. As seen above, the automaton $\mathcal{A}[\alpha]$ may be nondeterministic even when $\mathcal{A}$ is a DFA. In this section we study

the size of the minimal DFA that recognizes $L(\mathcal{A}[\alpha])$. We show that it may actually be exponentially larger than $\mathcal{A}$. Formally, we have the following.

Theorem 1. *Abstraction of DFAs may involve an exponential blow-up.*

Proof. We describe a family $\mathcal{A}_1, \mathcal{A}_2, \ldots$ of DFAs and corresponding abstraction functions $\alpha_1, \alpha_2, \ldots$, such that $\mathcal{A}_n$ has $O(n)$ states whereas the smallest DFA that recognizes $L(\mathcal{A}_n[\alpha_n])$ is exponential in n.

The DFA $\mathcal{A}_n$ (see Figure 1) consists of n chains, each having 5 states. Its alphabet is $\{1, \ldots, n, \#, \$\}$. The abstraction α_n maps the first state in all chains to one state and maps all other states to themselves. Thus, as shown in the figure, there is an abstract transition from the initial abstract state to itself. Clearly, the language $L(\mathcal{A}_n[\alpha_n])$ is

$$\mathcal{L}_n = \{\$^*\#x_1x_2\ldots x_m\#x : \; x_i \in \{1,..,n\} \text{ for all } 1 \leq i \leq m, \text{and } x \in \{x_1,..,x_m\}\}.$$

We claim that every DFA that recognizes $\mathcal{L}_n$ has at least 2^n states. Indeed, when a DFA for $\mathcal{L}_n$ reads the second $\#$, it has to remember the subset of $\{1, \ldots, n\}$ that was read so far. Intuitively, while in $\mathcal{A}_i$ the number of leading \$'s in the word direct the automaton as to which letter it should expect at the end of the word, the abstraction hides this number, forcing the abstract automaton to guess it, or, in the deterministic setting, to remember all letters read so far.

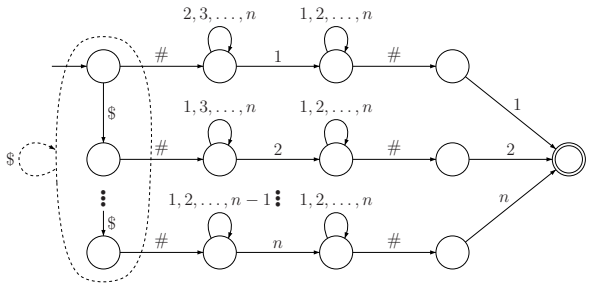

Figure 1: A description of the automata $\mathcal{A}_n$ and $\mathcal{A}_n[\alpha_n]$.

□

4. Deciding Whether Abstraction Helps

In this section we show that given a DFA and an abstraction function for it, the problem of deciding whether the abstraction is helpful is PSPACE-complete.

Formally, we define the decision problem AH (for *Abstraction Helps*), as follows.

Definition 1. Consider a DFA $\mathcal{A}$ with state space Q, and an abstraction function $\alpha : Q \to A$, where A is a set of abstract states. Let $\mathcal{D}$ be the minimal DFA such that $L(\mathcal{D}) = L(\mathcal{A}[\alpha])$. An input $\langle \mathcal{A}, \alpha \rangle$ is in AH iff the number of states in $\mathcal{D}$ is at most that in $\mathcal{A}$.

We start with the upper bound.

Theorem 2. *The problem AH is in PSPACE.*

Proof. Given $\mathcal{A}$ and α, let $\mathcal{D}[\alpha]$ be the DFA obtained by determining $\mathcal{A}[\alpha]$. The algorithm has to construct $\mathcal{D}[\alpha]$, minimize it, and compare the size of the obtained minimal automaton with the size of $\mathcal{A}$. Since the size of $\mathcal{D}[\alpha]$ is at most exponential in the size of $\mathcal{A}$, and it can be constructed and minimized on-the-fly, this can be done in PSPACE. □

In order to prove that AH is PSPACE-hard, we show a reduction from the decision problem ALL-Long-NFA: For an NFA $\mathcal{A}$ with n states, we say that $\mathcal{A} \in$ ALL-Long-NFA iff for all $w \in \Sigma^*$, if $|w| \geq n$ then $w \in L(\mathcal{A})$. The problem is similar to ALL-NFA – the problem of deciding whether a given NFA $\mathcal{A}$ is such that $L(\mathcal{A}) = \Sigma^*$. The latter is known to be PSPACE-hard and it is easy to reduce it to ALL-Long-NFA.

Theorem 3. *The problem AH is PSPACE-hard.*

Proof. We describe a reduction from ALL-Long-NFA to AH. Given an NFA $\mathcal{B}$, we construct a DFA $\mathcal{C}$ and an abstraction function α such that $\mathcal{B}$ is in ALL-Long-NFA iff $\langle \mathcal{C}, \alpha \rangle$ is in AH. Assume that $\mathcal{B}$ has n states, where we assume $n \geq 3$. We define $\mathcal{C}$ and α so that $\mathcal{C}[\alpha]$ simulates runs of three automata: the NFA $\mathcal{A}_{n-2}[\alpha_{n-2}]$, described in Theorem 1, the NFA $\mathcal{B}$, and an automaton CNT_n (for "count") that accepts words of length at least n. Let Σ_1 be the alphabet of $\mathcal{A}_{n-2}$ and Σ_2 be the alphabet of $\mathcal{B}$. Then, the alphabet of $\mathcal{C}$ and CNT_n is $\Sigma_1 \times \Sigma_2$. As detailed in the sequel, we construct $\mathcal{C}$ and α so that $\mathcal{C}$ has at least $n + 1$ states, and $\mathcal{C}[\alpha]$ accepts a word in Σ^* iff its length is at least n and either its projection on Σ_1 is in $\mathcal{A}_{n-2}[\alpha_{n-2}]$ or its projection on Σ_2 is in $L(\mathcal{B})$.

We first show that if we construct $\mathcal{C}$ and α as above, then $\mathcal{B} \in$ ALL-Long-NFA iff $\langle \mathcal{C}, \alpha \rangle \in$ AH. First, recall that if $\mathcal{B} \in$ ALL-Long-NFA, then $\mathcal{B}$ accepts all words of length at least n. Thus, by our construction, the language $L(\mathcal{C}[\alpha])$ is the set of words that are of length at least n. Since $L(\mathcal{C}[\alpha])$ can be recognized by a DFA with $n + 1$ states whereas $\mathcal{C}$ has at least $n + 1$ states, we conclude that $\langle \mathcal{C}, \alpha \rangle \in$ AH. For the other direction, if $\mathcal{B} \notin$ ALL-Long-NFA, there is a word of length at least n that $\mathcal{B}$ does not accept. Then, a DFA that recognizes $L(\mathcal{C}[\alpha])$ must, intuitively, distinguish between words in $L(\mathcal{A}_{n-2}[\alpha]) = \mathcal{L}_{n-2}$ and words that are not in $\mathcal{L}_{n-2}$.

Since a DFA that recognizes $\mathcal{L}_{n-2}$ has at least 2^{n-2} states, a DFA that recognizes $L(\mathcal{C}[\alpha])$ also has at least 2^{n-2} states. Thus, $\langle \mathcal{C}, \alpha \rangle \notin$ AH.

We continue to describe and prove the reduction formally. Recall that the input to ALL-Long-NFA is an NFA $\mathcal{B}$ and the input to the AH problem is a DFA $\mathcal{C}$ and an abstraction function α for it. We first replace the NFA $\mathcal{B}$ by a DFA $\mathcal{B}'$, which would be a component in the DFA $\mathcal{C}$. We do this by splitting states that have multiple outgoing transitions labeled with the same letter. Note that this naive determinization process changes the language of $\mathcal{B}$; thus $L(\mathcal{B}) \neq L(\mathcal{B}')$. This is still fine because we are going to define the abstraction function to map the different copies of the same state to the same abstract state, so $\mathcal{C}[\alpha]$ does simulate runs of the original automaton $\mathcal{B}$.

We proceed to define the automata in detail. Let $\mathcal{B} = \langle \Sigma_2, Q_\mathcal{B}, \delta_\mathcal{B}, q_0^\mathcal{B}, F_\mathcal{B} \rangle$ (for technical convenience, we assume that $\mathcal{B}$ has a single initial state). We construct the DFA $\mathcal{B}' = \langle \Sigma_2, Q_{\mathcal{B}'}, \delta_{\mathcal{B}'}, q_0^{\mathcal{B}'}, F_{\mathcal{B}'} \rangle$, where $Q_{\mathcal{B}'}$, $\delta_{\mathcal{B}'}$, $q_0^{\mathcal{B}'}$, and $F_{\mathcal{B}'}$ are defined as follows:

- For a state $q \in Q_\mathcal{B}$, we define the non-determinicity degree of q, denoted d_q, as $\max_{\sigma \in \Sigma_2} |\delta_\mathcal{B}(q, \sigma)|$. Then, $Q_{\mathcal{B}'} = \{\langle q, i \rangle : q \in Q_\mathcal{B}, i \in \{1, \ldots, d_q\}\}$. Note that $|Q_\mathcal{B}| \leq |Q_{\mathcal{B}'}| \leq |Q_\mathcal{B}|^2$.
- Consider a state $q \in Q_\mathcal{B}$ and a letter $\sigma \in \Sigma_2$. Let $\delta_\mathcal{B}(q, \sigma) = \{q_1, \ldots, q_k\}$, where the order of the q_i's is chosen arbitrarily. In $\mathcal{B}'$, the state q induces the k states $\langle q, 1 \rangle, \ldots, \langle q, k \rangle$. Since $k \leq d_q$, it is possible to define, for $1 \leq i \leq k$, the transition $\delta_{\mathcal{B}'}(\langle q, i \rangle, \sigma) = \langle q_i, 1 \rangle$.
- $q_0^{\mathcal{B}'} = \langle q_0^\mathcal{B}, 1 \rangle$.
- $F_{\mathcal{B}'} = \{\langle q, i \rangle : q \in F_\mathcal{B}, i \in \{1, \ldots, d_q\}\}$.

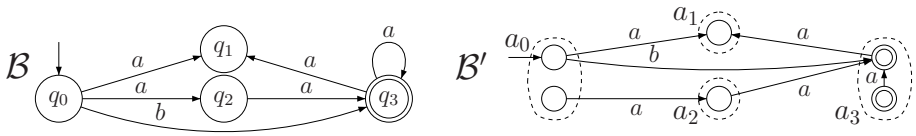

Figure 2: An example of constructing, given an NFA $\mathcal{B}$, a DFA $\mathcal{B}'$ and an abstraction function α_2.

Next, we define the abstraction function $\alpha_2 : Q_{B'} \to Q_\mathcal{B}$ by $\alpha_2(\langle q, i \rangle) = q$. It is easy to see that the abstracting automaton $\mathcal{B}'[\alpha_2]$ coincides with the NFA $\mathcal{B}$.

For example, consider the automata in Figure 2. The left automaton is the original NFA $\mathcal{B}$, the right automaton is the DFA $\mathcal{B}'$, and the dashed states in it represent the abstraction function α_2. Consider the state q_0 in $\mathcal{B}$. Since $d_{q_0} = 2$,

we split q_0 into two states. We define $\delta_{\mathcal{B}'}(\langle q_0, 1\rangle, b) = \langle q_3, 1\rangle$ and $\delta_{\mathcal{B}'}(\langle q_0, 2\rangle, a) = \langle q_2, 1\rangle$. Since $\alpha_2(\langle q_0, 1\rangle) = \alpha_2(\langle q_0, 2\rangle) = a_0$, we have that a_2 is a-successor of a_0 and a_3 is a b-successor of a_0, as in the original automaton.

Recall that $n = |Q_{\mathcal{B}}|$. Let $\mathcal{A}_{n-2} = \langle \Sigma_1, Q_{\mathcal{A}_{n-2}}, \delta_{\mathcal{A}_{n-2}}, q_0^{\mathcal{A}_{n-2}}, F_{\mathcal{A}_{n-2}}\rangle$ be the DFA described in the proof of Theorem 1, and let $\alpha_{n-2} : Q_{\mathcal{A}_{n-2}} \to Q'_{\mathcal{A}_{n-2}}$ be the abstraction function used there. Let $\Sigma = \Sigma_1 \times \Sigma_2$. We define the DFA $CNT_n = \langle \Sigma, Q_{CNT_n}, \delta_{CNT_n}, 0, \{n\}\rangle$, where $Q_{CNT_n} = \{0, \ldots, n\}$, for $0 \leq i \leq n-1$ and $\sigma \in \Sigma$, we have $\delta_{CNT_n}(i, \sigma) = i + 1$, and $\delta_{CNT_n}(n, \sigma) = n$. It is easy to see that $L(CNT_n) = \{w : |w| \geq n\}$.

We are now ready to define the DFA $\mathcal{C}$ and the abstraction function α for it. We define $\mathcal{C} = \langle \Sigma, Q_{\mathcal{A}_{n-2}} \times Q_{\mathcal{B}'} \times Q_{CNT_n}, \delta_{\mathcal{C}}, \langle q_0^{\mathcal{A}_{n-2}}, q_0^{\mathcal{B}'}, 0\rangle, F_{\mathcal{C}}\rangle$, where $F_{\mathcal{C}} = (Q_{\mathcal{A}_{n-2}} \times F_{\mathcal{B}'} \times \{n\}) \cup (F_{\mathcal{A}_{n-2}} \times Q_{\mathcal{B}'} \times \{n\})$, and $\delta_{\mathcal{C}}$ is defined as follows. For $\langle \sigma_1, \sigma_2\rangle \in \Sigma$ and $\langle a, b, i\rangle \in Q_{\mathcal{A}_{n-2}} \times Q_{\mathcal{B}'} \times Q_{CNT_n}$, we define $\delta_{\mathcal{C}}(\langle a, b, i\rangle, \langle \sigma_1, \sigma_2\rangle) = \langle \delta_{\mathcal{A}_{n-2}}(a, \sigma_1), \delta_{\mathcal{B}'}(b, \sigma_2), \delta_{CNT_n}(i, \langle \sigma_1, \sigma_2\rangle)\rangle$. Note that since $\mathcal{B}'$, $\mathcal{A}_{n-2}$, and CNT_n are deterministic, so is $\mathcal{C}$. Note also that $\mathcal{C}$ has at least $n + 1$ states, as required.

We define the abstraction function $\alpha : (Q_{\mathcal{A}_{n-2}} \times Q_{\mathcal{B}'} \times Q_{CNT_n}) \to (Q'_{\mathcal{A}_{n-2}} \times Q_{\mathcal{B}} \times Q_{CNT_n})$ by $\alpha(\langle a, b, i\rangle) = \langle \alpha_{n-2}(a), \alpha_2(b), i\rangle$.

Consider a word $w \in (\Sigma_1 \times \Sigma_2)^* = \Sigma^*$. Let $w_1 \in \Sigma_1^*$ and $w_2 \in \Sigma_2^*$ be w's projection on Σ_1 and Σ_2. We prove that $w \in L(\mathcal{C}[\alpha])$ iff $w \in L(CNT_n)$, and $w_1 \in L(\mathcal{A}_{n-2}[\alpha_{n-2}])$ or $w_2 \in L(\mathcal{B})$.

Consider a run r of $\mathcal{C}[\alpha]$ on w. Let r_1, r_2, and r_3 be the projections of r on $Q'_{\mathcal{A}_{n-2}}$, $Q_{\mathcal{B}}$, and Q_{CNT_n}, respectively. We claim that r is a legal run of $\mathcal{C}[\alpha]$ on w iff r_1, r_2, and r_3 are legal runs on the words w_1, w_2, and w, respectively. That is, $\mathcal{C}[\alpha]$ simulates runs of $\mathcal{A}_{n-2}[\alpha_{n-2}]$, $\mathcal{B}$, and CNT_n.

We start with the initial states. By the definition of $\mathcal{C}[\alpha]$, its initial state is $\alpha(\langle q_0^{\mathcal{A}_{n-2}}, q_0^{\mathcal{B}'}, 0\rangle) = \langle \alpha_{n-2}(q_0^{\mathcal{A}_{n-2}}), \alpha_2(q_0^{\mathcal{B}'}), 0\rangle$. The states $\alpha_{n-2}(q_0^{\mathcal{A}_{n-2}})$, $\alpha_2(q_0^{\mathcal{B}'})$, and 0 are initial in $\mathcal{A}_{n-2}[\alpha_{n-2}]$, $\mathcal{B}$, and CNT_n, respectively.

Consider two abstract states $a_{\mathcal{C}[\alpha]} = \langle a, b, i\rangle$ and $a'_{\mathcal{C}[\alpha]} = \langle a', b', i'\rangle$ in $\mathcal{C}[\alpha]$ and a letter $\sigma = \langle \sigma_1, \sigma_2\rangle \in \Sigma$. We claim that $a'_{\mathcal{C}[\alpha]}$ is a σ-successor of $a_{\mathcal{C}[\alpha]}$ iff a' is a σ_1-successor of a in $\mathcal{A}_{n-2}[\alpha_{n-2}]$, b' is a σ_2-successor of b in $\mathcal{B}$, and i' is a σ-successor of i in CNT_n.

Recall that $a'_{\mathcal{C}[\alpha]}$ is a σ-successor of $a_{\mathcal{C}[\alpha]}$ iff there are concrete states $c \in a_{\mathcal{C}[\alpha]}$ and $c' \in a'_{\mathcal{C}[\alpha]}$ such that c' is a σ-successor of c. By the definition of α, we have that $c = \langle c_a, c_b, i\rangle$ and $c = \langle c'_a, c'_b, i'\rangle$, where $c_a \in a$, $c_b \in b$, $c'_a \in a'$ and $c'_b \in b'$. By the definition of the transition function $\delta_{\mathcal{C}}$, we have that c' is a σ-successor of c iff c'_a is a σ_1-successor of c_a, c'_b is a σ_2-successor of c_b, and i' is a σ-successor of i. Thus, $\alpha_{n-2}(c'_a) = a'$ is a σ_1-successor of $\alpha_{n-2}(c_a) = a$, $\alpha_2(c'_b) = b'$ is a σ_2-successor of $\alpha_2(c_b) = b$, and i' is a σ-successor of i, and we are done.

Consider an index $0 \leq j \leq k$, and let $r_j = \langle a, b, i \rangle$ and $r_{j+1} = \langle a', b', i' \rangle$. By the above, r_{j+1} is a σ_{j+1}-successor of r_j iff a' is a σ^1_{j+1}-successor of a in $\mathcal{A}_{n-2}[\alpha_{n-2}]$, b' is a σ^2_{j+1}-successor of b in $\mathcal{B}$, and i' is a σ_{j+1}-successor of i in CNT_n. We conclude that r is a legal run of $\mathcal{C}[\alpha]$ on w iff r_1, r_2, and r_3 are legal runs of $\mathcal{A}_{n-2}[\alpha_{n-2}]$, $\mathcal{B}$, and CNT_n on w_1, w_2, and w, respectively.

Finally, we continue to prove that r is accepting in $\mathcal{C}[\alpha]$ iff r_3 is accepting in CNT_n, and r_1 is accepting in $\mathcal{A}_{n-2}[\alpha_{n-2}]$ or r_2 is accepting in $\mathcal{B}$. We prove the first direction and the second is dual.

By definition, $F'_{\mathcal{C}} = \{\alpha(q) : q \in F_{\mathcal{C}}\}$. We claim that $F'_{\mathcal{C}}$ is a union of two sets: $F'_1 = Q'_{\mathcal{A}_{n-2}} \times F_{\mathcal{B}} \times \{n\}$ and $F'_2 = F'_{\mathcal{A}_{n-2}} \times Q_{\mathcal{B}} \times \{n\}$. Recall that $F_{\mathcal{C}}$ is a union of two sets of states: $F_1 = Q_{\mathcal{A}_{n-2}} \times F_{\mathcal{B}'} \times \{n\}$ and $F_2 = F_{\mathcal{A}_{n-2}} \times Q_{\mathcal{B}'} \times \{n\}$. Since a state $\langle a, i \rangle \in F_{\mathcal{B}'}$ iff $a \in F_{\mathcal{B}}$, it holds that $\alpha(F_1) = F'_1$. Since the accepting states of $\mathcal{A}_{n-2}$ are mapped to themselves, it holds that $\alpha(F_2) = F'_2$.

If r is accepting, its last state is in F'_1 or F'_2. In both cases, r_3 is accepting since it ends in the state n. In the first case, r_1 is accepting since it ends in a state in $F'_{\mathcal{A}_{n-2}}$, and in the second case r_3 is accepting since it ends in $F_{\mathcal{B}}$, and we are done.

To conclude the proof of the theorem, we claim that $\mathcal{B} \in$ ALL-Long-NFA iff $\langle \mathcal{C}, \alpha \rangle \in$ AH. Recall that an automaton $\mathcal{B}$ of size n is in ALL-Long-NFA iff for every word $w \in \Sigma_2^*$, if $|w| \geq n$ then $w \in L(\mathcal{B})$. Also, recall that $\langle \mathcal{C}, \alpha \rangle \in$ AH iff the smallest DFA that accepts $L(\mathcal{C}[\alpha])$ is at most the size of $\mathcal{C}$.

For the first direction, assume that $\mathcal{B} \in$ ALL-Long-NFA. We claim that then, $L(\mathcal{C}[\alpha]) = \{w \in \Sigma^* : |w| \geq n\}$. Consider a word $w \in \Sigma^*$ and let w_1 be its projection on Σ_1. If $|w| < n$, then $w \notin L(CNT_n)$ and thus, by the simulation proven above, we have that $w \notin L(\mathcal{C}[\alpha])$. If $|w| \geq n$, then $w \in L(CNT_n)$. Since $\mathcal{B} \in$ ALL-Long-NFA, we know that $w_1 \in L(\mathcal{B})$. Thus, by the simulation proven above, $w \in L(\mathcal{C}[\alpha])$, and we are done. To conclude this direction, since $L(\mathcal{C}[\alpha])$ is the set of words over Σ that are longer than n, there is a deterministic automaton that accepts $L(\mathcal{C}[\alpha])$ with $n+1$ states. Namely, the automaton CNT_n, which has $n+1$ states. Since the size of $\mathcal{C}$ is at least $n+1$, we are done.

For the second direction, assume that $\mathcal{B} \notin$ ALL-Long-NFA. We claim that in this case there is no DFA that has less than 2^{n-2} states that recognizes $L(\mathcal{C}[\alpha])$. Assume by way of contradiction that $\mathcal{D}$ is such a DFA. Since $\mathcal{B} \notin$ ALL-Long-NFA, there is a word $w_2 = w_1^2 \dots w_m^2 \in \Sigma_2^*$ such that $|w_2| = m \geq n$ and $w_2 \notin L(\mathcal{B})$.

Consider the words over Σ_1 that are of length $m-2$, have a (possibly empty) prefix of \$'s, followed by the letter $\#$, followed by a string that contains letters from a subset of $\{1, \dots, n-2\}$. Denote this set of words by W_{n-2}. For example, if $n = 5$ and $m = 6$, then $\$\#12, \$\$\#1, \#123 \in W_{n-2}$. Since $m > n-2$, all subsets of $\{1, \dots, n-2\}$ appear in W_{n-2}, and since every word in W_{n-2} corresponds to a single subset of $\{1, \dots, n-2\}$, we have $|W_{n-2}| = 2^{n-2}$. Consider the runs of

$\mathcal{D}$ on words whose projection on Σ_1 is in W_{n-2}. Since there are less than 2^{n-2} states in $\mathcal{D}$, there are two different such words for which the run of $\mathcal{D}$ reaches the same state. Denote their projection on Σ_1 by u and v, and their matching subsets of $\{1, \ldots, n-2\}$, by U and V, respectively.

As in similar proofs that are based on the language $\mathcal{A}_{n-2}[\alpha_{n-2}]$, we can use a letter in the symmetric difference of U and V in order to fool $\mathcal{D}$ and get a contradiction. Formally, since $U \neq V$ there is, without loss of generality, a letter $i \in U \setminus V$. Consider the two words: $w_1 = (u\#i) \otimes w_2$ and $w_1' = (v\#i) \otimes w_2$, where for $a = a_1 \ldots a_k \in \Sigma_1^k$ and $b = b_1 \ldots b_k \in \Sigma_2^k$ we define $a \otimes b = \langle a_1, b_1 \rangle, \ldots, \langle a_k, b_k \rangle \in \Sigma^k$. Note that since the length w_2 is m, and the lengths of u and v is $m-2$, w_1 and w_1' are well defined.

Clearly, $w_1 \in L(\mathcal{C}[\alpha])$, as its length is $m \geq n$ and, since $i \in U$, its projection on Σ_1, namely $u\#i$, is in $L(\mathcal{A}_{n-2}[\alpha_{n-2}])$. However, $w_1' \notin L(\mathcal{C}[\alpha])$. Indeed, since $i \notin V$, its projection on Σ_1, namely $v\#i$, is not in $L(\mathcal{A}_{n-2}[\alpha_{n-2}])$, and its projection on Σ_2, namely w_2, is not in $L(\mathcal{B})$. On the other hand, $\mathcal{D}$ accepts w_1 iff it accepts w_1', contradicting the fact $L(\mathcal{D}) = L(\mathcal{C}[\alpha])$.

We conclude that every deterministic automaton that recognizes $L(\mathcal{C}[\alpha])$ must have at least 2^{n-2} states. Since the size of $\mathcal{C}$ is at most $n^2 \cdot 5(n-2) \cdot (n+1)$, we are done. □

References

[1] E. Clarke, O. Grumberg, D. Peled, Model Checking, MIT Press, 1999.

[2] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement for symbolic model checking, Journal of the ACM 50 (2003) 752–794.

[3] A. Pnueli, R. Rosner, On the synthesis of a reactive module, in: Proc. 16th ACM Symp. on Principles of Programming Languages, pp. 179–190.

[4] M. Vardi, Automatic verification of probabilistic concurrent finite-state programs, in: Proc. 26th IEEE Symp. on Foundations of Computer Science, pp. 327–338.

[5] O. Kupferman, Y. Lustig, What triggers a behavior?, in: Proc. 7th Int. Conf. on Formal Methods in Computer-Aided Design, IEEE Computer Society, 2007, pp. 146–153.

[6] O. Kupferman, M. Vardi, Memoryful branching-time logics, in: Proc. 21st IEEE Symp. on Logic in Computer Science, pp. 265–274.