RAIRO-Theor. Inf. Appl. 43 (2009) 69–94
DOI: 10.1051/ita:2007061

Available online at:
www.rairo-ita.org

HYPER-MINIMIZING MINIMIZED DETERMINISTIC FINITE STATE AUTOMATA *

Andrew Badr[1], Viliam Geffert[2] and Ian Shipman[3]

Abstract. We present the first (polynomial-time) algorithm for reducing a given deterministic finite state automaton (DFA) into a *hyper-minimized* DFA, which may have fewer states than the classically minimized DFA. The price we pay is that the language recognized by the new machine can differ from the original on a finite number of inputs. These hyper-minimized automata are optimal, in the sense that every DFA with fewer states must disagree on infinitely many inputs. With small modifications, the construction works also for finite state transducers producing outputs. Within a class of finitely differing languages, the hyper-minimized automaton is not necessarily unique. There may exist several non-isomorphic machines using the minimum number of states, each accepting a separate language finitely-different from the original one. We will show that there are large structural similarities among all these smallest automata.

Mathematics Subject Classification. 68Q70.

1. Introduction

Automata theory is one of the classic parts of theoretical computer science, and also a common first step in the study of this field. A deterministic one-way finite state automaton (DFA) is the fundamental standard model for regular languages. At first glance, all important problems for this simplest model were already solved several decades ago.

Keywords and phrases. Finite state automata, regular languages.

* *Supported by the Slovak Grant Agency for Science (VEGA) under contract "Combinatorial Structures and Complexity of Algorithms", and by the Science and Technology Assistance Agency under contract APVT-20-004104.*

[1] 3210 Acklen Ave., Nashville, TN 37212, USA; badr@uiuc.edu

[2] Department of Computer Science, P. J. Šafárik University, Jesenná 5, 04001 Košice, Slovakia; viliam.geffert@upjs.sk

[3] Department of Mathematics, University of Chicago, 5734 S. University Ave., Chicago, IL 60637, USA; ics@math.uchicago.edu

Article published by EDP Sciences

The first classic result of this kind, solved more than 50 years ago [11,13], is the algorithm for minimization of the number of states in a given deterministic finite state automaton. It is known that the automaton produced by this algorithm cannot be beaten by any other automaton: each automaton M' with fewer states must disagree in acceptance *for at least one input*. Moreover, the smallest possible machine is unique, up to isomorphism. All these results have become a part of many textbooks. (See *e.g.* [8].)

But if we allow the language to change by some finite number of words, further minimization is possible. We present here a simple polynomial-time algorithm that reduces the given deterministic finite state automaton into a *hyper-minimized* automaton, which may have fewer states than the classically minimized automaton. The new machine is *almost-equivalent* to the original machine: it may differ in acceptance on a finite number of input strings[1].

In addition, we shall show that our automaton is indeed a smallest one with this property, even after allowing additional finite-differences from the original language. That is, every DFA with fewer states than our hyper-minimized automaton must disagree in acceptance not only for a single input, as in the classical case, but *for infinitely many inputs*.

With minor updates, the algorithm works for finite state transducers as well. We can construct hyper-minimized almost-equivalent automata that do not recognize languages, but produce some outputs. An almost-equivalent transducer must produce the correct sequence of output symbols *after an initial "probation period" of fixed length*. The length of this fixed period does not depend on the length of the input. Actually, this is exactly what people expect from automata in most real life applications. Even very large and complex systems are quite often made up of a large number of diverse small units, interrelated only by their inputs and outputs. As a typical example, a manufacturer installing a new unit into a production line will always use a probation period for testing, during which nobody expects an absolutely exact behavior. This gives an additional motivation for studying almost-equivalent simulations on almost-equivalent machines.

The hyper-minimized automaton is not necessarily unique. We can find examples in which the given minimized n-state acceptor M can be replaced by several different hyper-minimized machines, all of them using the same number of states, below n, but each accepting a different variation of the original language $L(M)$. There is one important special case: the hyper-minimized automaton is always unique for machines using a unary input alphabet.

Even in the general case, there are large structural similarities among all these machines. The state set of any DFA can be divided into two parts: *1 preamble* states – reachable only by a finite number of inputs from the initial state and *2 kernel* states – reachable by infinitely many inputs. (There are no unreachable

[1]The idea of using an almost-equivalent simulation to obtain an almost-equivalent machine has already appeared, *e.g.*, in [7], for an efficient simulation of two-way nondeterministic unary automata on all but finitely many inputs.

states in a hyper-minimized automaton.) The preamble parts of any two almost-equivalent hyper-minimized machines are isomorphic, except for the acceptance-values of states. Their kernels are isomorphic in the standard way, preserving also the acceptance-values. Finally, the isomorphism preserves the initial states if they are in the preambles. (This might not be the case for machines with empty preambles.) The machines can also differ in some of the transitions that go from preamble states to kernel states.

We shall also provide a simple tool for testing hyper-minimality: an automaton M is hyper-minimized if and only if, in M, *1* there does not exist an unreachable state, *2* there does not exist a pair of different but equivalent states, and *3* there does not exist a pair of different but *almost-equivalent* states, such that at least one of them is a preamble state. (Two states are almost-equivalent if there exist at most finitely many input strings α such that, starting from these two states and reading α in parallel, we can reach a state pair with different acceptance-values.) Note that the first two conditions alone characterize minimized automata. (All three conditions can be efficiently tested.)

Finally, there is no upper bound on the number of states we can save by hyper-minimization. For each $m \leq n$, there exists a regular language L such that the unique minimized acceptor recognizing L uses exactly n states, but for which the hyper-minimized acceptor recognizing a finitely-different language (also unique in this case) uses exactly m states.

2. Elementary properties

We begin with presenting some basic notation on finite state automata. For a more detailed exposition, the reader is referred, *e.g.*, to [8,10]. We also introduce some new elementary definitions and properties required later.

Definition 2.1. A *deterministic finite state automaton* is a quintuple $M = (Q, \Sigma, \delta, q_{\mathrm{I}}, F)$, where Q denotes a finite set of states, Σ a finite set of input symbols, $\delta : Q \times \Sigma \to Q$ a transition function, $q_{\mathrm{I}} \in Q$ an initial state, and $F \subseteq Q$ a set of accepting (or final) states.

The function δ can be extended to $\delta : Q \times \Sigma^* \to Q$ in the standard way, so that $\delta(q, \alpha_1\alpha_2) = \delta(\delta(q, \alpha_1), \alpha_2)$, for each $q \in Q$ and $\alpha_1, \alpha_2 \in \Sigma^*$.

An input string $\alpha \in \Sigma^*$ is *accepted*, if $\delta(q_{\mathrm{I}}, \alpha) \in F$. The language consisting of all accepted strings will be denoted by $L(M)$.

Definition 2.2. Two states q_{A} and q_{B} are *equivalent*, which we will denote by $q_{\mathrm{A}} \equiv q_{\mathrm{B}}$, if, for each $\alpha \in \Sigma^*$, we have $\delta(q_{\mathrm{A}}, \alpha) \in F$ if and only if $\delta(q_{\mathrm{B}}, \alpha) \in F$.

Two states q_{A} and q_{B} are *almost-equivalent*, denoted by $q_{\mathrm{A}} \sim q_{\mathrm{B}}$, if there exists a $k \geq 0$ such that, for each $\alpha \in \Sigma^*$ of length $|\alpha| \geq k$, we have $\delta(q_{\mathrm{A}}, \alpha) \in F$ if and only if $\delta(q_{\mathrm{B}}, \alpha) \in F$.

In other words, equivalent states q_{A}, q_{B} have no distinguishing input string α such that the descendant states $\delta(q_{\mathrm{A}}, \alpha)$ and $\delta(q_{\mathrm{B}}, \alpha)$ have different acceptance-values.

Similarly, for almost-equivalent states q_A, q_B, the acceptance-values of the descendants $\delta(q_A, \alpha)$ and $\delta(q_B, \alpha)$ are the same for all but finitely many values of α.

The above definition can easily be extended to pairs of states belonging to different machines, but using the same input alphabet.

Given two automata $M_1 = (Q_1, \Sigma, \delta_1, q_{I,1}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{I,2}, F_2)$, the states $q_A, q_B \in Q_1 \cup Q_2$ are equivalent, if they are equivalent in the automaton $M = (Q, \Sigma, \delta, q_I, F)$, where $Q = Q_1 \cup Q_2$, $\delta(q, a) = \delta_1(q, a)$ if $q \in Q_1$, but $\delta(q, a) = \delta_2(q, a)$ if $q \in Q_2$, and $F = F_1 \cup F_2$. (Here we assume, without loss of generality, that $Q_1 \cap Q_2 = \emptyset$.) The initial state q_I can be chosen arbitrarily. The same extension is used to define almost-equivalent states, as well.

Definition 2.3. (*a*) Two automata $M_1 = (Q_1, \Sigma, \delta_1, q_{I,1}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{I,2}, F_2)$ are said to be *equivalent*, denoted by $M_1 \equiv M_2$, if their initial states $q_{I,1}$ and $q_{I,2}$ are equivalent, *i.e.*, if $q_{I,1} \equiv q_{I,2}$. Similarly, M_1 and M_2 are *almost-equivalent*, denoted by $M_1 \sim M_2$, if $q_{I,1} \sim q_{I,2}$.

(*b*) Two languages L_1 and L_2 are *finitely-different*, if their symmetric difference is finite, which will be denoted by $L_1 \sim L_2$.

In other words, equivalent machines agree in acceptance for all inputs, and hence they accept the same language. That is, $L(M_1) = L(M_2)$. Similarly, almost-equivalent machines agree in acceptance for all but finitely many inputs, and hence $L(M_1) \sim L(M_2)$.

Definition 2.4. The automaton $M = (Q, \Sigma, \delta, q_I, F)$ is said to be *minimized*, if no automaton $M' = (Q', \Sigma, \delta', q'_I, F')$ with fewer states is equivalent to M.

The automaton M is *hyper-minimized*, if no automaton M' with fewer states is almost-equivalent to M.

That is, a minimized M cannot be replaced by a machine M' using a smaller number of states or else M and M' disagree in acceptance for at least one input string $\alpha \in \Sigma^*$. If M is hyper-minimized, then each M' with fewer states must disagree in acceptance not only for a single input, but for infinitely many inputs as well.

Before proceeding, we shall present some simple properties of equivalent states, which can be found in many standard textbooks concerning the finite state automata. (See *e.g.* [8].) Then, these properties will be compared to the corresponding ones for almost-equivalent states.

Where statements presented in Facts 2.5–2.7 and Lemmas 2.8–2.10 refer to q_A or q_B, these are states either from the same DFA, or from two automata sharing the same input alphabet.

Fact 2.5. *"$\equiv$" is an equivalence relation, i.e., reflexive, symmetric, and transitive.*

Fact 2.6. *If $q_A \equiv q_B$, then for each $\alpha \in \Sigma^*$, $\delta(q_A, \alpha) \equiv \delta(q_B, \alpha)$.*

Fact 2.7. *$q_A \equiv q_B$ if and only if*

(*a*) *for each $a \in \Sigma$, $\delta(q_A, a) \equiv \delta(q_B, a)$, and*

(*b*) *$q_A \in F$ if and only if $q_B \in F$.*

The corresponding properties for almost-equivalent states look as follows:

Lemma 2.8. *"$\sim$" is an equivalence relation.*

Proof. The argument is an easy consequence of Definition 2.2. For example, if $q_A \sim q_B$, then we have some $k_1 \geq 0$ such that, for each $\alpha \in \Sigma^*$ of length $|\alpha| \geq k_1$, the acceptance-values of the states $\delta(q_A, \alpha)$ and $\delta(q_B, \alpha)$ are the same. Similarly, if $q_B \sim q_C$, we have some $k_2 \geq 0$ such that, for each $\alpha \in \Sigma^*$ of length $|\alpha| \geq k_2$, the acceptance-values of $\delta(q_B, \alpha)$ and $\delta(q_C, \alpha)$ are also the same. This gives that $\delta(q_A, \alpha) \in F$ if and only if $\delta(q_C, \alpha) \in F$, for each $\alpha \in \Sigma^*$ of length $|\alpha| \geq \max\{k_1, k_2\}$. Thus, $q_A \sim q_C$, if $q_A \sim q_B$ and $q_B \sim q_C$.

The arguments for symmetry and reflexivity are trivial. □

Lemma 2.9. *If $q_A \sim q_B$, then for each $\alpha \in \Sigma^*$, $\delta(q_A, \alpha) \sim \delta(q_B, \alpha)$.*

Proof. Suppose that, for some $\tilde{\alpha} \in \Sigma^*$, the state $q'_A = \delta(q_A, \tilde{\alpha})$ is *not* almost-equivalent to $q'_B = \delta(q_B, \tilde{\alpha})$. Then there must exist infinitely many inputs $\alpha_1, \alpha_2, \ldots$ such that, for each $i = 1, 2, \ldots$, the acceptance-values of the descendants $\delta(q'_A, \alpha_i)$ and $\delta(q'_B, \alpha_i)$ are different. That is, $\delta(q'_A, \alpha_i) \in F$ if and only if $\delta(q'_B, \alpha_i) \notin F$. Now, by taking the sequence $\tilde{\alpha}\alpha_1, \tilde{\alpha}\alpha_2, \ldots$, we get that $\delta(q_A, \tilde{\alpha}\alpha_i) \in F$ if and only if $\delta(q_B, \tilde{\alpha}\alpha_i) \notin F$. Thus, we have obtained an infinite sequence of inputs giving different acceptance-values for the descendants of q_A and q_B, and hence these two states are not almost-equivalent.

Therefore, if $\delta(q_A, \tilde{\alpha}) \not\sim \delta(q_B, \tilde{\alpha})$, for some $\tilde{\alpha} \in \Sigma^*$, then $q_A \not\sim q_B$. □

Lemma 2.10. *$q_A \sim q_B$ if and only if*

(a) for each $a \in \Sigma$, $\delta(q_A, a) \sim \delta(q_B, a)$.

Proof. The argument for the "⇒" part is a special case of Lemma 2.9, using $|\alpha| = 1$.

For the converse implication, we use the fact that the number of symbols in the input alphabet is finite, *i.e.*, $\Sigma = \{a_1, a_2, \ldots, a_m\}$. Now assume that, for each symbol $a_i \in \Sigma$, $\delta(q_A, a_i) \sim \delta(q_B, a_i)$. Therefore, we have some $k_i \geq 0$ such that, for each $\alpha \in \Sigma^*$ of length $|\alpha| \geq k_i$, the acceptance-values of the states $\delta(\delta(q_A, a_i), \alpha)$ and $\delta(\delta(q_B, a_i), \alpha)$ are the same.

This gives, for each $\alpha' \in \Sigma^*$ of length $|\alpha'| \geq 1 + \max\{k_1, k_2, \ldots, k_m\}$, that $\delta(q_A, \alpha') \in F$ if and only if $\delta(q_B, \alpha') \in F$, and hence $q_A \sim q_B$. □

The reachability of states from the initial state plays an important role in constructing and understanding hyper-minimized automata:

Definition 2.11. Let $M = (Q, \Sigma, \delta, q_I, F)$ be a finite state automaton.

A state $q \in Q$ is *unreachable*, if there does not exist an input string $\alpha \in \Sigma^*$ such that $\delta(q_I, \alpha) = q$.

The state $q \in Q$ is in the *preamble* if there exists at least one input string $\alpha \in \Sigma^*$ with $\delta(q_I, \alpha) = q$, but the number of such strings is finite.

The state $q \in Q$ is in the *kernel* if there exist infinitely many (all different) input strings $\alpha \in \Sigma^*$ with $\delta(q_I, \alpha) = q$.

The respective sets of unreachable, preamble, and kernel states will be denoted by U, P, and K.

The next two lemmas present some elementary properties that will be used for an efficient identification of kernel and removable states.

Lemma 2.12. *For each automaton $M = (Q, \Sigma, \delta, q_{\text{I}}, F)$, a state $q \in Q$ is in the kernel if and only if it can be reached from the initial state by a path passing through a state q' such that there exists a cycle beginning and ending in q'.*

Proof. Clearly, if q can be reached by a path passing through some q', such that there exists a cycle beginning and ending in q', then q can be reached by infinitely many inputs, obtained by iterating this cycle arbitrarily many times. Thus, q is in the kernel.

Conversely, since a kernel state q can be reached from q_{I} by infinitely many inputs, we have $q = \delta(q_{\text{I}}, \alpha)$ also for some $\alpha \in \Sigma^*$ of length $|\alpha| \geq \|Q\|$. But then some state q' must be repeated in the course of reading α, which gives the cycle containing the state q'. □

Lemma 2.13. *Let $M = (Q, \Sigma, \delta, q_{\text{I}}, F)$ be a DFA with no unreachable states. Then, for each $q_{\text{A}}, q_{\text{B}} \in Q$, the statement "at least one of the states q_{A}, q_{B} is in the preamble" is equivalent to the following:*

(I) *either q_{A} is in the preamble and q_{A} is not reachable from q_{B};*
(II) *or q_{B} is in the preamble and q_{B} is not reachable from q_{A}.*

Proof. The argument for the "⇐" part is trivial. Now assume that at least one of the states q_{A}, q_{B} is in the preamble.

First, if q_{A} is in the kernel, then q_{B} is in the preamble. Moreover, q_{B} is not reachable from q_{A} by any computation path, or else q_{B} is also in the kernel, being reachable from q_{I} by infinitely many inputs *via* q_{A}, which is a contradiction. Thus, we have obtained (II).

Similarly, if q_{B} is in the kernel, then q_{A} is in the preamble. Moreover, q_{A} is not reachable from q_{B}, or else q_{A} is also in the kernel, a contradiction. This gives (I).

Finally, if neither of the states q_{A}, q_{B} is in the kernel, they are both in the preamble. In addition, either q_{B} is not reachable from q_{A} or q_{A} is not reachable from q_{B}. Supposing the contrary, we get a cycle containing both q_{A} and q_{B}, reachable from the initial state q_{I}, since there are no unreachable states in M. But then both q_{A} and q_{B} are in the kernel, which is a contradiction. Thus, we have derived "(I) or (II)". □

The following lemma will be useful for simplification of a hyper-minimization algorithm.

Lemma 2.14. *Let $M = (Q, \Sigma, \delta, q_{\text{I}}, F)$ be a DFA with no unreachable states. Then, for each pair of distinct almost-equivalent states $q_{\text{A}} \sim q_{\text{B}}$,*

- *both q_{A} and q_{B} are in the preamble and, moreover, neither of them is reachable from the other one by any computation path; or*

- *there exists a state q_C in the kernel, such that q_C is almost-equivalent to q_A and q_B (including the possibility that $q_C = q_A$ or $q_C = q_B$).*

Proof. If at least one of the states q_A, q_B is in the kernel, we are done, this state can be used as the state q_C. So assume that both q_A and q_B are in the preamble. If neither of them is reachable from the other one by any computation path, we are done again.

Therefore, without loss of generality, assume now that both q_A and q_B are in the preamble, but there exists a computation path connecting q_A with q_B. That is, $\delta(q_A, \tilde{\alpha}) = q_B$, for some string $\tilde{\alpha} \in \Sigma^*$, $\tilde{\alpha} \neq \varepsilon$.

Now, let $q_1 = q_A$, and let $q_{k+1} = \delta(q_k, \tilde{\alpha})$, for each $k \geq 1$. It is easy to see that this gives a sequence of states $q_1, q_2, q_3, \ldots$ such that q_{k+1} is reachable from q_k. Clearly, such sequence must contain a kernel state, since some state must be repeated, *i.e.*, we must get $q_{i'} = q_{i''}$ for some $i' < i''$. But then $q_{i'}, \ldots, q_{i''}$ are kernel states lying along a cycle.

Since $q_1 = q_A$, it is sufficient to show that $q_1 \sim q_2 \sim q_3 \sim \ldots$. First, $q_1 = q_A \sim q_B = \delta(q_A, \tilde{\alpha}) = \delta(q_1, \tilde{\alpha}) = q_2$. Therefore, $q_1 \sim q_2$. By Lemma 2.9, using inductively $q_k \sim q_{k+1}$, we then get that $\delta(q_k, \tilde{\alpha}) \sim \delta(q_{k+1}, \tilde{\alpha})$, which gives $q_{k+1} \sim q_{k+2}$.

Thus, by taking $q_C = q_{i'}$, we get a kernel state almost-equivalent to $q_A = q_1$. □

3. Hyper-minimized acceptors

3.1. Characterization

Here we present the fundamental properties of hyper-minimized automata.

The following theorem will play an important role in our considerations. It says that descendants of two almost-equivalent states are not only almost-equivalent, as shown in Lemma 2.9, but that they become fully equivalent, if they are obtained by reading a sufficiently long input.

Theorem 3.1. *Let $M_1 = (Q_1, \Sigma, \delta_1, q_{I,1}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{I,2}, F_2)$ be two automata using the same input alphabet. Let $q_A \in Q_1$ and $q_B \in Q_2$.*

If $q_A \sim q_B$, then, for each $\alpha \in \Sigma^*$ of length $|\alpha| \geq \|Q_1\| \cdot \|Q_2\|$, we have $\delta_1(q_A, \alpha) \equiv \delta_2(q_B, \alpha)$.

Proof. Let q_A and q_B be two states satisfying the assumptions of the theorem. Suppose that, for some $\tilde{\alpha} \in \Sigma^*$ of length $|\tilde{\alpha}| \geq \|Q_1\| \cdot \|Q_2\|$, the state $q'_A = \delta_1(q_A, \tilde{\alpha})$ is *not* equivalent to $q'_B = \delta_2(q_B, \tilde{\alpha})$. Then there must exist at least one $\beta \in \Sigma^*$ such that the acceptance-values of the descendants $\delta_1(q'_A, \beta)$ and $\delta_2(q'_B, \beta)$ are different. Without loss of generality, assume that $\delta_1(q'_A, \beta) \in F_1$ but $\delta_2(q'_B, \beta) \notin F_2$. Therefore, $\delta_1(q_A, \tilde{\alpha}\beta) \in F_1$ and, at the same time, $\delta_2(q_B, \tilde{\alpha}\beta) \notin F_2$.

But, by assumption, $\tilde{\alpha} \in \Sigma^*$ is sufficiently long so that it can be expressed in the form $\tilde{\alpha} = a_1 a_2 \ldots a_\ell$, for some symbols $a_1, a_2, \ldots, a_\ell \in \Sigma$, with $\ell \geq \|Q_1\| \cdot \|Q_2\|$. Now, consider the sequence of state pairs $\langle q_A^{(0)}, q_B^{(0)} \rangle, \langle q_A^{(1)}, q_B^{(1)} \rangle, \langle q_A^{(2)}, q_B^{(2)} \rangle, \ldots, \langle q_A^{(\ell)}, q_B^{(\ell)} \rangle$, where $q_A^{(i)} = \delta_1(q_A, a_1 \ldots a_i)$ and $q_B^{(i)} = \delta_2(q_B, a_1 \ldots a_i)$, for $i = 0, 1, 2, \ldots, \ell$.

As some special cases, we have $\langle q_{\rm A}^{(0)}, q_{\rm B}^{(0)}\rangle = \langle q_{\rm A}, q_{\rm B}\rangle$ and $\langle q_{\rm A}^{(\ell)}, q_{\rm B}^{(\ell)}\rangle = \langle q'_{\rm A}, q'_{\rm B}\rangle$. Note that this sequence contains more pairs than $\|Q_1\| \cdot \|Q_2\|$, but each pair $\langle q_{\rm A}^{(i)}, q_{\rm B}^{(i)}\rangle$ is in $Q_1 \times Q_2$. Therefore, some pair must have been repeated, *i.e.*, we have $\langle q_{\rm A}^{(i')}, q_{\rm B}^{(i')}\rangle = \langle q_{\rm A}^{(i'')}, q_{\rm B}^{(i'')}\rangle$, for some i' and i'' satisfying $0 \le i' < i'' \le \ell$.

This divides the string $\tilde{\alpha}$ into $\tilde{\alpha} = \alpha_1\alpha_2\alpha_3$, where $\alpha_1 = a_1 \dots a_{i'}$, $\alpha_2 = a_{i'+1} \dots a_{i''}$, and $\alpha_3 = a_{i''+1} \dots a_\ell$, not excluding the possibility that $\alpha_1 = \varepsilon$ or $\alpha_3 = \varepsilon$, but always with $\alpha_2 \neq \varepsilon$, such that

- $\delta_1(q_{\rm A}, \alpha_1) = q_{\rm A}^{(i')}$, $\delta_1(q_{\rm A}^{(i')}, \alpha_2) = q_{\rm A}^{(i'')} = q_{\rm A}^{(i')}$, $\delta_1(q_{\rm A}^{(i')}, \alpha_3) = q'_{\rm A}$, and $\delta_1(q'_{\rm A}, \beta) \in F_1$,
- $\delta_2(q_{\rm B}, \alpha_1) = q_{\rm B}^{(i')}$, $\delta_2(q_{\rm B}^{(i')}, \alpha_2) = q_{\rm B}^{(i'')} = q_{\rm B}^{(i')}$, $\delta_2(q_{\rm B}^{(i')}, \alpha_3) = q'_{\rm B}$, and $\delta_2(q'_{\rm B}, \beta) \notin F_2$.

By iterating the string α_2 arbitrarily many times, we thus obtain an infinite sequence of inputs, namely, $\alpha_1\alpha_2\alpha_3\beta$, $\alpha_1\alpha_2\alpha_2\alpha_3\beta$, $\alpha_1\alpha_2\alpha_2\alpha_2\alpha_3\beta, \dots$, such that each of them gives different acceptance-values for the descendants of $q_{\rm A}$ and $q_{\rm B}$, and hence these two states are not almost-equivalent.

Therefore, if $\delta_1(q_{\rm A}, \tilde{\alpha}) \not\equiv \delta_2(q_{\rm B}, \tilde{\alpha})$, for some $\tilde{\alpha} \in \Sigma^*$ of length $|\tilde{\alpha}| \ge \|Q_1\| \cdot \|Q_2\|$, then $q_{\rm A} \not\sim q_{\rm B}$. □

By an easy modification, the above theorem can be updated for state pairs belonging to the same automaton M.

Now we are ready to turn our attention to the properties of hyper-minimized automata.

Theorem 3.2. *Let $M = (Q, \Sigma, \delta, q_{\rm I}, F)$ be a finite state acceptor. If, in M,*

(*a*) *there exists an unreachable state $q_{\rm A}$; or*
(*b*) *there exists a pair of different but equivalent states $q_{\rm A} \equiv q_{\rm B}$; or*
(*c*) *there exists a pair of different but almost-equivalent states $q_{\rm A} \sim q_{\rm B}$, such that at least one of them is in the preamble,*

then M can be replaced by an almost-equivalent $M' = (Q', \Sigma, \delta', q'_{\rm I}, F')$ using fewer states.

Proof. If the condition (a) or (b) is satisfied, it is well-known that M can be replaced by a fully equivalent M' (hence, also almost-equivalent M') with fewer states.

So assume that (c) holds true, but (a) does not hold, *i.e.*, all states are reachable from $q_{\rm I}$. By Lemma 2.13, we get that

(I) either $q_{\rm A}$ is in the preamble and, moreover, $q_{\rm A}$ is not reachable from $q_{\rm B}$;
(II) or $q_{\rm B}$ is in the preamble and, moreover, $q_{\rm B}$ is not reachable from $q_{\rm A}$.

Consider now the case (I), that is, assume that the state $q_{\rm A}$ is in the preamble, and that it is not reachable from $q_{\rm B}$ by any computation path. (The state reduction and argument for the case (II) are symmetrical, just swapping the roles of $q_{\rm A}$ and $q_{\rm B}$.)

Then M' is obtained by "redirecting" all edges ending in q_A from q_A to q_B. More precisely, $M' = (Q\setminus\{q_A\}, \Sigma, \delta', q'_I, F\setminus\{q_A\})$, where

$$\delta'(q,a) = \begin{cases} \delta(q,a), & \text{if } \delta(q,a) \neq q_A, \\ q_B, & \text{if } \delta(q,a) = q_A, \end{cases}$$

for each $q \in Q\setminus\{q_A\}$ and $a \in \Sigma$. Similarly,

$$q'_I = \begin{cases} q_I, & \text{if } q_I \neq q_A, \\ q_B, & \text{if } q_I = q_A. \end{cases}$$

Clearly, M' uses a smaller number of states than does M. It only remains to show that M and M' are almost-equivalent.

It is easy to see that there exists a constant $k_1 \geq 0$ such that, if $q_A = \delta(q_I, \beta)$, for some $\beta \in \Sigma^*$, then $|\beta| \leq k_1$. (Recall that q_A can be reached in M only by a finite number of input strings, so k_1 is simply the length of the longest one.)

Since $q_A \sim q_B$ in M, there also exists a constant $k_2 \geq 0$ such that, for each $\beta \in \Sigma^*$ of length $|\beta| \geq k_2$, we have $\delta(q_A, \beta) \in F$ if and only if $\delta(q_B, \beta) \in F$.

Consider now an input $\alpha \in \Sigma^*$ of length $|\alpha| \geq k_1 + k_2$. If, in M, the path examining α does not visit the state q_A, the same transitions are used also in M'. Therefore, M and M' agree in acceptance for all inputs, provided that the examining paths do not visit q_A in M.

Now assume that the path examining α does visit the state q_A in M. This cannot happen more than once along any path, or else we would have a cycle beginning and ending in q_A. But then we would have infinitely many inputs ending in q_A, which is a contradiction, since q_A is in the preamble.

Recall that $|\alpha| \geq k_1 + k_2$. Therefore, α can be decomposed into $\alpha = \alpha_1\alpha_2$, such that $\delta(q_I, \alpha_1) = q_A$, which gives that $|\alpha_1| \leq k_1$, and hence also that $|\alpha_2| \geq k_2$. Note also that $\alpha \in L(M)$ if and only if $\delta(q_A, \alpha_2) \in F$.

In M', we have $\delta'(q'_I, \alpha_1) = q_B$ (not excluding the possibility that $q'_I = q_B$ and $\alpha_1 = \varepsilon$), and hence $\alpha \in L(M')$ if and only if $\delta'(q_B, \alpha_2) \in F\setminus\{q_A\}$. Using the definition of δ' and the fact that, by (I), the state q_A is not reachable from q_B by any computation path in M, we get that $\alpha \in L(M')$ if and only if $\delta(q_B, \alpha_2) \in F$.

But $|\alpha_2| \geq k_2$, and hence $\delta(q_B, \alpha_2) \in F$ if and only if $\delta(q_A, \alpha_2) \in F$. Summing up, $\alpha \in L(M)$ if and only if $\alpha \in L(M')$. Therefore, M and M' agree in acceptance for all inputs α of length $|\alpha| \geq k_1 + k_2$. □

It should be pointed out that the above state reduction, redirecting all edges ending in q_A from q_A to q_B, can also be used in the case (b). For a pair of equivalent states $q_A \equiv q_B$, the above reduction is symmetrical; it does not matter whether we redirect from q_A to q_B or vice versa.

However, in the case of almost-equivalent states $q_A \sim q_B$, such that $q_A \not\equiv q_B$, we can redirect only in accordance with the conditions presented by the items (I) and (II) of Lemma 2.13 and Theorem 3.2; from q_A to q_B, if the item (I) is satisfied; from q_B to q_A, if the item (II) is satisfied. The reduction is symmetrical only if both q_A and q_B are in the preamble and, moreover, neither of them is reachable

from the other one by any path. When both these states are in the kernel, we cannot use this reduction at all.

As an example, consider the states $G \sim H$ in the automaton presented by Figure 2 below. Both these states are in the preamble. First, we can safely redirect from G to H. The new machine will disagree only on finitely many inputs, of length at most 4. However, if we redirected from H to G, the new automaton would accept all inputs in the form $b^k aa$, for each $k \geq 5$, but these words are rejected by the original machine. Now we are going to prove the converse statement for Theorem 3.2.

Theorem 3.3. *Let $M = (Q, \Sigma, \delta, q_{\text{I}}, F)$ be a finite state acceptor. If M can be replaced by an almost-equivalent $M' = (Q', \Sigma, \delta', q'_{\text{I}}, F')$ using fewer states, then, in M,*

- *(a) there exists an unreachable state q_{A}; or*
- *(b) there exists a pair of different but equivalent states $q_{\text{A}} \equiv q_{\text{B}}$; or*
- *(c) there exists a pair of different but almost-equivalent states $q_{\text{A}} \sim q_{\text{B}}$, such that at least one of them is in the preamble.*

Proof. If there is an unreachable state in M, we are done, the condition (a) has been satisfied. Therefore, assume that all states are reachable from the initial state q_{I} in M.

Since the state set Q is finite, it can be expressed in the form $Q = \{q_1, q_2, \ldots, q_m\}$, with $q_{\text{I}} = q_1$. All these states are reachable, and hence we can find a sequence of input strings $w_1, w_2, \ldots, w_m$ such that, for each i, $q_i = \delta(q_{\text{I}}, w_i)$. However, these strings will be chosen a little bit more carefully:

- if q_i is in the preamble, let w_i be a shortest string satisfying $q_i = \delta(q_{\text{I}}, w_i)$;
- if q_i is in the kernel, let w_i be a shortest string of length $|w_i| \geq \|Q\|^2$ satisfying $q_i = \delta(q_{\text{I}}, w_i)$. (Such string must exist, since q_i is reachable by infinitely many different input strings.)

Using these inputs in the automaton M', we get a sequence of states $r_1, r_2, \ldots, r_m \in Q'$ such that, for each i, $r_i = \delta'(q'_{\text{I}}, w_i)$.

Now we are going to utilize the fact that M and M' are almost-equivalent, and hence $q_{\text{I}} \sim q'_{\text{I}}$. Since $q_i = \delta(q_{\text{I}}, w_i)$ and $r_i = \delta'(q'_{\text{I}}, w_i)$, we have, by Lemma 2.9, that $q_i \sim r_i$, for each $i = 1, 2, \ldots, m$.

But, by assumption of the theorem, M' has fewer states than M, and hence $\|Q'\| < \|Q\| = m$. Therefore, the sequence $r_1, r_2, \ldots, r_m \in Q'$ must repeat some states, *i.e.*, we have $r_{i'} = r_{i''}$ for some i' and i'' satisfying $1 \leq i' < i'' \leq m$.

This gives $q_{i'} \sim r_{i'} = r_{i''} \sim q_{i''}$, that is, $q_{i'} \sim q_{i''}$. Thus, if at least one of these two states is in the preamble, we are done, the condition (c) has been satisfied. Therefore, assume now that both $q_{i'}$ and $q_{i''}$ are in the kernel.

First, recall that $q_{i'} = \delta(q_{\text{I}}, w_{i'})$, $r_{i'} = \delta'(q'_{\text{I}}, w_{i'})$, and $q_{\text{I}} \sim q'_{\text{I}}$. In addition, since $q_{i'}$ is in the kernel, the string $w_{i'}$ is of length $|w_{i'}| \geq \|Q\|^2 > \|Q\| \cdot \|Q'\|$. But then, by Theorem 3.1, we have that $q_{i'} \equiv r_{i'}$.

Second, by the same reasoning, using the fact that $q_{i''}$ is also in the kernel and hence the string $w_{i''}$ is also of length $|w_{i''}| > \|Q\| \cdot \|Q'\|$, we get that $q_{i''} \equiv r_{i''}$.

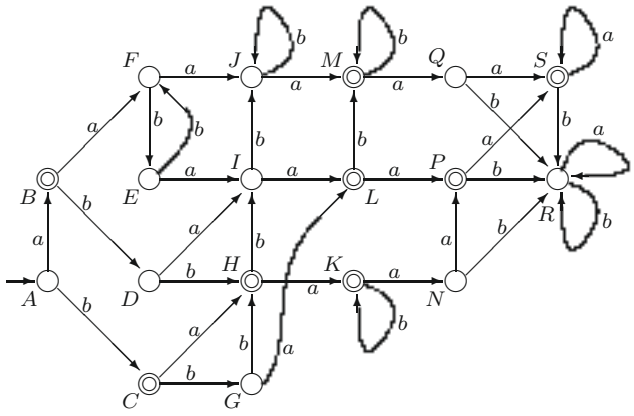

FIGURE 1. An example of an automaton. By grouping the states into classes of equivalent states and these, in turn, into classes of almost-equivalent states, we get the following partition: $\{\{A\}\}, \{\{B\}\}, \{\{C\}, \{D\}\}, \{\{E\}\}, \{\{F\}\}, \{\{G\}, \{H\}, \{I\}, \{J\}\}$, $\{\{K, M\}, \{L\}\}, \{\{N, Q\}, \{P, S\}\}, \{\{R\}\}$.

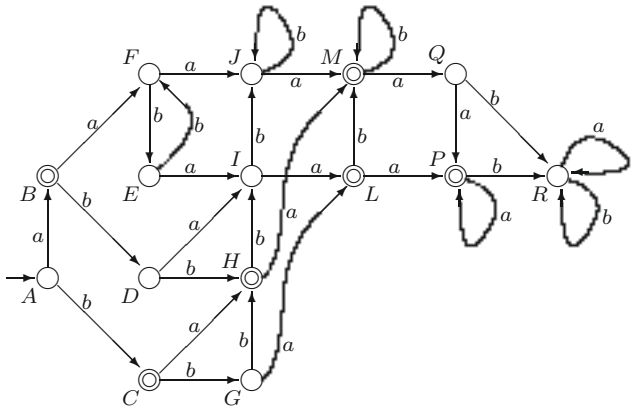

FIGURE 2. Minimized version equivalent to the original (unique), obtained by removing equivalent states and redirecting edges ending in these states. More precisely, we have redirected $K \Rightarrow M$, $N \Rightarrow Q$, and $S \Rightarrow P$. Now the partition into classes of almost-equivalent states is: $\{A\}$, $\{B\}$, $\{C, D\}$, $\{E^*\}$, $\{F^*\}$, $\{G, H, I^*, J^*\}$, $\{M^*, L^*\}$, $\{Q^*, P^*\}$, $\{R^*\}$. Kernel states are marked by asterisk.

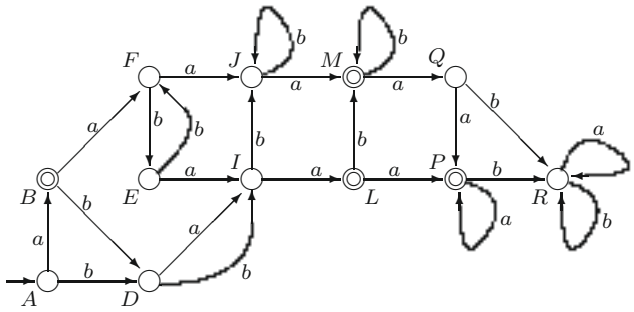

FIGURE 3. One of possible hyper-minimized versions, accepting a language finitely-different from the original, obtained by the reductions $C \Rightarrow D$ and $G, H \Rightarrow I^*$. This results in the following differences: $L' \setminus L = \{abbaa, baaa\}$ and $L \setminus L' = \{abb, abbbaa, b, ba, babaa, bbb, bbbbaa\}$. Another possible solution can be obtained, *e.g*, by changing $\delta(D, b)$ from I to J. Such machine will correctly reject *abbaa*, but makes a new mistake by rejecting *bbaa*.

But then $q_{i'} \equiv r_{i'} = r_{i''} \equiv q_{i''}$, that is, $q_{i'} \equiv q_{i''}$. Now we are done, the condition (b) has been satisfied. □

By combining Theorems 3.2 and 3.3 we thus get:

Theorem 3.4 (characterization of hyper-minimized DFA's)**.** *A finite state acceptor $M = (Q, \Sigma, \delta, q_{\mathrm{I}}, F)$ is hyper-minimized, i.e., it cannot be replaced by an almost-equivalent M' using fewer states, if and only if, in M,*

(*a*) *there does not exist an unreachable state q_{A}; and*
(*b*) *there does not exist a pair of different but equivalent states $q_{\mathrm{A}} \equiv q_{\mathrm{B}}$; and*
(*c*) *there does not exist a pair of different but almost-equivalent states $q_{\mathrm{A}} \sim q_{\mathrm{B}}$, such that at least one of them is in the preamble.*

The above theorem should be compared with the classical characterization of minimized automata:

Fact 3.5. *A finite state acceptor $M = (Q, \Sigma, \delta, q_{\mathrm{I}}, F)$ is minimized, i.e., it cannot be replaced by an equivalent M' using fewer states, if and only if, in M,*

(*a*) *there does not exist an unreachable state q_{A}; and*
(*b*) *there does not exist a pair of different but equivalent states $q_{\mathrm{A}} \equiv q_{\mathrm{B}}$.*

3.2. Structural similarities

It is well-known that, up to isomorphism, the minimized acceptor is unique for each regular language. This can be formulated as follows.

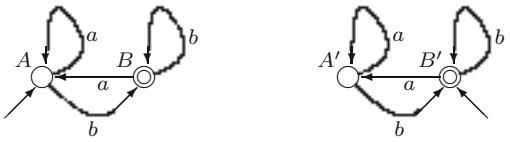

FIGURE 4. An example of two hyper-minimized almost-equivalent automata without any preambles yet not isomorphic. The machines do not agree in acceptance of ε. The only possible mapping of states that preserves accepting states does not preserve initial states.

Fact 3.6. *If two minimized acceptors $M = (Q, \Sigma, \delta, q_{\text{I}}, F)$ and $M' = (Q', \Sigma, \delta', q'_{\text{I}}, F')$ are equivalent, then they are isomorphic, i.e., there exists a function $h\colon Q \to Q'$ with the following properties.*

(*a*) *The function is a bijection, i.e.,*
- *if $q_{\text{A}} \neq q_{\text{B}}$, then $h(q_{\text{A}}) \neq h(q_{\text{B}})$, for each $q_{\text{A}}, q_{\text{B}} \in Q$;*
- *for each $q' \in Q'$, there exists a $q \in Q$ such that $h(q) = q'$.*

(*b*) *The function preserves the machine's behavior, i.e.,*
- *$h(q_{\text{I}}) = q'_{\text{I}}$,*
- *$h(\delta(q, a)) = \delta'(h(q), a)$, for each $q \in Q$ and $a \in \Sigma$,*
- *$h(q) \in F'$ if and only if $q \in F$, for each $q \in Q$.*

The converse of Fact 3.6 holds as well: two isomorphic machines are equivalent.

The corresponding properties for hyper-minimized machines are not so perfectly beautiful. We can find examples in which the given minimized n-state automaton M can be replaced by several different hyper-minimized machines, all of them using the same number of states, below n, but each accepting a separate language finitely-different from the original $L(M)$. (See the example shown in Figs. 1–3. Below Fig. 3, you will find a hint saying how to obtain another hyper-minimized automaton accepting another finitely different language.) Nevertheless, such machines can differ only in *1* edges leaving the preamble for the kernel, *2* acceptance-values for states in the preamble, or *3* initial states, if the preamble is empty, not containing any state. (Such an example is shown in Fig. 4.)

Roughly speaking, the kernels of two almost-equivalent hyper-minimized machines are isomorphic in the standard way and their preambles are also isomorphic, except for acceptance values.

The next theorem introduces such mapping.

Theorem 3.7 (morphism keeping almost-equivalence)**.** *For each two almost-equivalent DFA's $M = (Q, \Sigma, \delta, q_{\text{I}}, F)$ and $M' = (Q', \Sigma, \delta', q'_{\text{I}}, F')$, there exists a function $h\colon Q \to Q'$ with the following properties.*

(*a*) *If $q \in P$, then $q \sim h(q)$. If $q \in K$, then $q \equiv h(q)$.*
(*b*) *If $q \in K$, then $h(q) \in K'$.*

(Here P, P' denote the respective sets of preamble states in M, M' and, similarly, K, K' their respective sets of kernel states.)

Proof. The argument uses the mapping presented already in the proof of Theorem 3.3. Let $Q_{\mathrm{R}} = \{q_1, q_2, \ldots, q_m\}$ be the set of all reachable states in M, with $q_{\mathrm{I}} = q_1$. Now we can find a sequence of inputs $w_1, w_2, \ldots, w_m$ satisfying $q_i = \delta(q_{\mathrm{I}}, w_i)$, for each $i = 1, 2, \ldots, m$. To avoid any ambiguity, these strings are chosen as follows:

- If q_i is in P, the preamble of M, let w_i be the first string such that $q_i = \delta(q_{\mathrm{I}}, w_i)$.
- If q_i is in K, the kernel of M, let w_i be the first string of length $|w_i| \geq \|Q\| \cdot \|Q'\|$ satisfying $q_i = \delta(q_{\mathrm{I}}, w_i)$.

In "our" string ordering, a shorter string always precedes the longer one, and strings of equal length go in lexicographic order. (For example, bb < aba < bba.) Now we are ready to introduce the function $h\colon Q \to Q'$. Let

- $h(q_i) = \delta'(q'_{\mathrm{I}}, w_i)$, for $i = 1, 2, \ldots, m$;
- $h(q) = q'_{\mathrm{I}}$, for each unreachable state $q \in Q$.

It only remains to prove (a) and (b).

First, for each $q_i \in P$, using Lemma 2.9, we get $q_i \sim h(q_i)$, since $q_i = \delta(q_{\mathrm{I}}, w_i)$, $h(q_i) = \delta'(q'_{\mathrm{I}}, w_i)$, and $q_{\mathrm{I}} \sim q'_{\mathrm{I}}$. Similarly, for each $q_i \in K$, using the fact that then $|w_i| \geq \|Q\| \cdot \|Q'\|$, we get that $q_i \equiv h(q_i)$, by Theorem 3.1.

Finally, if $q_i \in K$, then $h(q_i) = \delta'(q'_{\mathrm{I}}, w_i)$ is reachable from the initial state in M' by a string of length $|w_i| \geq \|Q\| \|Q'\| \geq \|Q'\|$. Therefore, the computation path reading w_i must repeat some states, which gives, by Lemma 2.12, that $h(q_i) \in K'$. □

If, moreover, the above two automata are classically minimized, then the function h becomes a classical isomorphism between their kernel parts.

Theorem 3.8 (kernel isomorphism)**.** *For each two minimized almost-equivalent DFA's $M = (Q, \Sigma, \delta, q_{\mathrm{I}}, F)$ and $M' = (Q', \Sigma, \delta', q'_{\mathrm{I}}, F')$, there exists a function $h\colon Q \to Q'$ satisfying, besides the conditions (a) and (b) of Theorem 3.7, also the following conditions.*

(c) *Between the kernel parts, the function is a bijection.*
  - *If $q \in K$, then $h(q) \in K'$.*
  - *For each $q_{\mathrm{A}}, q_{\mathrm{B}} \in K$, $q_{\mathrm{A}} \neq q_{\mathrm{B}}$ implies that $h(q_{\mathrm{A}}) \neq h(q_{\mathrm{B}})$.*
  - *For each $q' \in K'$, there exists a $q \in K$ such that $h(q) = q'$.*

(d) *In the kernels, the function preserves the machine's behavior, except for the initial states.*
  - *For each $q \in K$ and $a \in \Sigma$, $h(\delta(q, a)) = \delta'(h(q), a)$.*
  - *For each $q \in K$, $h(q) \in F'$ if and only if $q \in F$.*

Proof. The function h can be obtained by the use of Theorem 3.7. We only have to show that it satisfies the conditions (c) and (d).

First, in Theorem 3.7, we already proved that $h(q) \in K'$ if $q \in K$. Now we shall show that

(i) if $q_{\mathrm{A}} \neq q_{\mathrm{B}}$, then $h(q_{\mathrm{A}}) \neq h(q_{\mathrm{B}})$, for each $q_{\mathrm{A}}, q_{\mathrm{B}} \in K$.

Suppose that $h(q_{\mathrm{A}}) = h(q_{\mathrm{B}})$. Then, by (a) in Theorem 3.7, we get that $q_{\mathrm{A}} \equiv h(q_{\mathrm{A}}) = h(q_{\mathrm{B}}) \equiv q_{\mathrm{B}}$, that is, $q_{\mathrm{A}} \equiv q_{\mathrm{B}}$. By Fact 3.5, the automaton M, being

minimized, does not have a pair of different but equivalent states. Thus, $q_A = q_B$. Therefore, $q_A \neq q_B$ implies $h(q_A) \neq h(q_B)$, for each $q_A, q_B \in K$.

As a straightforward consequence, by combining (i) with (b) in Theorem 3.7, we easily get that $\|K\| \leq \|K'\|$. However, all reasonings above were symmetrical: by swapping the roles of M and M', we can obtain a dual function $\hat{h}\colon Q' \to Q$ with the analogous properties, which results in $\|K'\| \leq \|K\|$. Therefore, $\|K\| = \|K'\|$.

By combining (b) in Theorem 3.7 with (i) and $\|K\| = \|K'\|$, we get that, for each $q' \in K'$, there must exist a $q \in K$ such that $h(q) = q'$. This completes the argument for the item (c).

Recall that, by (a) in Theorem 3.7, $q \equiv h(q)$, for each state $q \in K$. But then $\delta(q, a) \equiv \delta'(h(q), a)$, for each symbol $a \in \Sigma$, using Fact 2.7. Clearly, if $q \in K$, then also $\delta(q, a) \in K$, and hence $\delta(q, a) \equiv h(\delta(q, a))$, using (a) in Theorem 3.7 again. Combining these facts together, we obtain that $h(\delta(q, a)) \equiv \delta'(h(q), a)$.

But M' is minimized and hence, in M', there does not exist a pair of different but equivalent states, by Fact 3.5. Therefore, $h(\delta(q, a)) = \delta'(h(q), a)$.

Finally, for each $q \in K$, $q \equiv h(q)$, by (a) in Theorem 3.7, and hence $q \in F$ if and only if $h(q) \in F'$, by Fact 2.7. This completes the argument for the item (d). □

Finally, the highest degree of similarity can be found between two almost-equivalent hyper-minimized machines. Besides isomorphic kernels, their preambles are isomorphic as well, except for acceptance-values in preambles.

Theorem 3.9 (preamble isomorphism)**.** *For each two hyper-minimized almost-equivalent DFA's $M = (Q, \Sigma, \delta, q_I, F)$ and $M' = (Q', \Sigma, \delta', q'_I, F')$, there exists a function $h\colon Q \to Q'$ satisfying, besides the conditions (a)–(d) of Theorems 3.7 and 3.8, also the following conditions.*

(e) *Between the preamble parts, the function is a bijection.*
- *If $q \in P$, then $h(q) \in P'$.*
- *For each $q_A, q_B \in P$, $q_A \neq q_B$ implies that $h(q_A) \neq h(q_B)$.*
- *For each $q' \in P'$, there exists a $q \in P$ such that $h(q) = q'$.*

(f) *In the preambles, the function preserves the machine's behavior, except for acceptance and edges leaving the preamble.*
- *If $q_I \in P$, then $h(q_I) = q'_I$.*
- *For each $q \in P$ and $a \in \Sigma$ satisfying $\delta(q, a) \in P$, $h(\delta(q, a)) = \delta'(h(q), a)$.*

Proof. The function h can again be obtained by the use of Theorem 3.7. By Theorems 3.7 and 3.8, we already have an argument for (a)–(d), since hyper-minimized automata are also minimized. It only remains to prove (e) and (f).

First, we need a little stronger statement than the corresponding claim in (e):

(i) if $q_A \neq q_B$, then $h(q_A) \neq h(q_B)$, for each $q_A, q_B \in Q$.

Supposing that $h(q_A) = h(q_B)$, we get, by (a) in Theorem 3.7, that $q_A \sim h(q_A) = h(q_B) \sim q_B$, which gives $q_A \sim q_B$. By Theorem 3.4, the hyper-minimized automaton M does not have a pair of different but almost-equivalent states, such that at least one of them is in the preamble. Thus, if at least one of the states q_A, q_B is in the preamble, we have $q_A = q_B$. If both these states are in the kernel,

we get $q_A = q_B$ by the use of (c) in Theorem 3.8. Therefore, if $q_A \neq q_B$, then $h(q_A) \neq h(q_B)$, for each $q_A, q_B \in Q$, and hence also for each $q_A, q_B \in P$.

Using (c) in Theorem 3.8, we can also conclude that if $q \notin K$, then $h(q) \notin K'$. Otherwise, for $h(q) = q' \in K'$, we get $h(q) = q' = h(q'')$, for some $q'' \in K$. Since $q \notin K$ but $q'' \in K$, we have $q \neq q''$ together with $h(q) = h(q'')$. But this contradicts (I). Therefore, if $q \in P$, then $h(q) \in P'$.

As a straightforward consequence of (I), we have $\|Q\| \leq \|Q'\|$. Since all reasonings above were symmetrical, we get also $\|Q'\| \leq \|Q\|$, by swapping the roles of M and M'. Therefore, $\|Q\| = \|Q'\|$. But then, using $\|K\| = \|K'\|$ shown in Theorem 3.8, we have $\|P\| = \|P'\|$.

By combining $\|Q\| = \|Q'\|$ with (I), we can easily see that, for each $q' \in Q'$, there must exist a $q \in Q$ such that $h(q) = q'$. Moreover, if $q' \in P'$, then $q \in P$. Otherwise we would have $q \in K$, which would imply, by (b) in Theorem 3.7, that $h(q) = q' \in K'$, a contradiction. Therefore, for each $q' \in P'$, there must exist a $q \in P$ such that $h(q) = q'$. This completes the argument for the item (e).

Now, assume that $q_I \in P$. By (a) in Theorem 3.7, $q_I \sim h(q_I)$. In addition, $q_I \sim q'_I$, since M and M' are almost-equivalent machines. Therefore, $q'_I \sim h(q_I)$.

Using (e), we also get that $h(q_I) \in P'$. But M' is hyper-minimized an hence, in M', there does not exist a pair of different but almost-equivalent states, such that at least one of them is in the preamble, by Theorem 3.4. Therefore, if $q'_I \sim h(q_I)$ and $h(q_I) \in P'$, then $q'_I = h(q_I)$.

Recall that, by (a) in Theorem 3.7, $q \sim h(q)$, for each state $q \in P$. But then $\delta(q, a) \sim \delta'(h(q), a)$, for each symbol $a \in \Sigma$, by Lemma 2.10. Now assume that $\delta(q, a) \in P$. This gives $\delta(q, a) \sim h(\delta(q, a))$, using (a) in Theorem 3.7 again. Combining these facts together, we obtain that $h(\delta(q, a)) \sim \delta'(h(q), a)$.

Finally, using (e), we see that if $\delta(q, a) \in P$, then $h(\delta(q, a)) \in P'$. But M' is hyper-minimized and hence, if $h(\delta(q, a)) \sim \delta'(h(q), a)$ and $h(\delta(q, a)) \in P'$, then $h(\delta(q, a)) = \delta'(h(q), a)$. This holds for each $q \in P$ and $a \in \Sigma$ satisfying $\delta(q, a) \in P$. This completes the argument for the item (f). □

Remark 3.10. *The converse of Theorem 3.9 does not hold in general. With empty preambles, two hyper-minimized automata can recognize infinitely different languages and, at the same time, they can have a function $h\colon Q \to Q'$ satisfying the conditions (a)–(f) of Theorems 3.7–3.9. (Consequently, the same applies to Ths. 3.7 and 3.8.)*

Moreover, there exist hyper-minimized automata with nonempty isomorphic preambles and, at the same time, nonempty isomorphic kernels, for some function h satisfying the conditions (b)–(f), such that they differ in acceptance on infinitely many inputs.

Proof. First, consider the following two unary languages: $L_1 = \{b^k : k \text{ is even}\}$ and $L_2 = \{b^k : k \text{ is odd}\}$. Clearly, these two languages are infinitely different. The unique minimized automata for L_1 and L_2 use the same cycle counting modulo two, the only difference is in their initial states. By Corollary 3.11, shown below, these two machines are already hyper-minimized. The only mapping preserving acceptance satisfies also the conditions (a)–(f). Since there are no states in the preambles, several conditions are satisfied automatically.

Now consider the following two binary languages: $L'_1 = \{ab^k : k \text{ is even}\}$ and $L'_2 = \{ab^k : k \text{ is odd}\}$. Let M_1 and M_2 be the respective unique minimized automata for L'_1 and L'_2. Each of these machines uses one state in the preamble, namely, the initial state, and three states in the kernel: two states in a loop counting b's modulo 2, and a single-state loop discarding inputs containing two or more a's or beginning with b.

These machines are already hyper-minimized. (Hint: using ab^i or b^i, for $i \geq 0$, we get two infinite distinguishing sequences of inputs, suitable for each pair of states in M_1 and M_2. We leave the details to the reader.)

It is also quite easy to find a mapping h satisfying the conditions (b)–(f). Therefore, the preamble as well as the kernel parts of M_1 and M_2 are isomorphic, but the languages differ on infinitely many strings. □

As already pointed out earlier, a hyper-minimized machine M is also minimized, for the language $L(M)$. (Compare Th. 3.4 with Fact 3.5.) The converse statement does not hold in general, since a minimized M can potentially be replaced by a new machine with even fewer states, accepting a language finitely-different from $L(M)$. (For examples, see Figs. 1–3, or Th. 3.13.)

In many cases, the following simple statement can be used to guarantee that the given machine is not only minimized, but also hyper-minimized.

Corollary 3.11. *Let $M = (Q, \Sigma, \delta, q_1, F)$ be a minimized DFA with a cycle beginning and ending in its initial state. Then M is already hyper-minimized.*

Proof. Since, in M, there exists a cycle beginning and ending in the initial state, there are no states in the preamble. Therefore, M satisfies the conditions (a)–(c) of Theorem 3.4. □

3.3. Unary automata

Now we shall focus our attention on *unary* regular languages, *i.e.*, on automata with a single letter input alphabet. Unary (tally) languages play an important role in theoretical computer science, as languages with a very low information content. Many of their properties are quite different from the general or binary case. (See, *e.g.*, [2,4–7,12].)

To simplify the notation, we fix $\Sigma = \{1\}$ and present a single-step transition $\delta(q_A, 1) = q_B$ in the form $q_A \xrightarrow{1} q_B$.

It is obvious that the transition function of a unary automaton is determined by two quantities, the length of the initial segment and the length of the subsequent loop. Thus, unless there are some unreachable states, such automaton consists of

- an initial preamble $p_s \xrightarrow{1} p_{s-1} \xrightarrow{1} p_{s-2} \ \ldots \ p_1 \xrightarrow{1} q_0$, reading the first s symbols, where p_s is the initial state, and q_0 the first state of the subsequent loop;
- a kernel loop $q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \ \ldots \ q_{\ell-1} \xrightarrow{1} q_0$, counting modulo ℓ,

for some $s \geq 0$ and $\ell \geq 1$. If already the initial state is a part of the kernel loop, the length of the initial preamble is zero. Recall that two machines of the same preamble and kernel loop lengths can still differ in the distribution of their accepting states.

Though, in general, there may exist several different hyper-minimized almost-equivalent machines, the hyper-minimized acceptor is always unique in the unary case, up to isomorphism.

Theorem 3.12. *Let $M = (Q, \{1\}, \delta, q_{\text{I}}, F)$ and $M' = (Q', \{1\}, \delta', q'_{\text{I}}, F')$ be two almost-equivalent hyper-minimized finite state automata using a unary input alphabet. Then these two automata are equivalent and minimized (and hence isomorphic).*

Proof. First, since M is hyper-minimized, it is also minimized. Second, it is easy to see that the length of its initial preamble is zero. Otherwise, we would be able to remove the preamble states $p_s, p_{s-1}, \ldots, p_1$ and use the loop state $q_{(-s) \bmod \ell}$ as a new initial state. Clearly, the new machine would use fewer states and would agree with M in acceptance on all inputs of length above $s-1$. But this contradicts the fact that M is hyper-minimized. Therefore, M consists of a single cycle of length ℓ containing all states, beginning and ending in q_{I}.

By the same argument, M' is also minimized and consists of a single cycle of length ℓ', beginning and ending in q'_{I}. It is also easy to see that $\ell' = \ell$ or else one of the machines M, M' would not be hyper-minimized.

It only remains to show that these two automata are fully equivalent. Suppose, for contradiction, that there exists a $k \geq 0$ such that the input 1^k is accepted by M but rejected by M'. Now, by taking the sequence $1^{k+1\cdot\ell}, 1^{k+2\cdot\ell}, 1^{k+3\cdot\ell}, \ldots$, we get infinitely many inputs accepted by M but rejected by M'. But this contradicts the assumption that M and M' are almost-equivalent. Therefore, they must be equivalent.

The existence of an isomorphic mapping $h\colon Q \to Q'$ follows from Fact 3.6. □

3.4. Descriptional complexity

Clearly, if a minimized automaton uses some n states, then its hyper-minimized almost-equivalent counterpart(s) will use some $m \leq n$ states. A natural question arising here is whether there exists some upper bound on the number of states we can save by allowing a potentially different acceptance for a finite number of inputs. The answer is negative. We can actually show the complete state hierarchy for this relation, using even unary witness languages.

Theorem 3.13. *Let $m \leq n$ be two positive integers. Then there exists a (unary) regular language L such that the unique minimized acceptor recognizing L uses exactly n states, but the unique hyper-minimized acceptor recognizing a language finitely-different from L uses exactly m states.*

Proof. Assume first that $m < n$. Let $s = n-m$. Now, consider an automaton M consisting of

- an initial preamble $p_s \xrightarrow{1} p_{s-1} \xrightarrow{1} p_{s-2} \ \ldots\ p_1 \xrightarrow{1} q_0$, reading the first s symbols, where p_s is the initial state, and q_0 the first state of the subsequent loop;
- a kernel loop $q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \ \ldots\ q_{m-1} \xrightarrow{1} q_0$, counting modulo m;
- finally, the only accepting state is q_{m-1}.

It can be easily seen that M accepts the language $L = \{1^{(s+m-1)+k\cdot m} : k \geq 0\}$, using exactly $s+m = n$ states. This number cannot be reduced, since the shortest string in L is of length $s+m-1 = n-1$. (It should be clear that each automaton with fewer states must accept an input shorter than $n-1$. The only exception is an automaton accepting the empty set.) Therefore, M is minimized.

Now, remove the preamble states $p_s, p_{s-1}, \ldots, p_1$ and make the new initial state from the state $q_{(-s) \bmod m}$. This gives a new automaton M' using exactly m states and accepting the language $L' = \{1^k : k \bmod m = (s+m-1) \bmod m\}$. Clearly, L' is only finitely different from L and hence M' is almost-equivalent to M. Second, there are neither unreachable states nor pairs of equivalent states in M', since M' consists of a single cycle in which only a single state is accepting. Therefore, M' is the unique minimized acceptor for L'. Finally, there is a cycle beginning and ending in the initial state of M' which gives, by Corollary 3.11, that M' is also hyper-minimized. That is, no additional finite-difference from L saves a single state. This completes the argument for $m < n$.

If $m = n$, we can use the language $L'' = \{1^k : k \bmod n = 0\}$. It is obvious that the unique minimized automaton accepting L'' consists of a single cycle counting modulo n, where the only accepting state is the initial state. This automaton is also hyper-minimized, by Corollary 3.11, and hence M coincides with M' here. □

4. Algorithm for hyper-minimization of DFA's

Now we are ready to present an algorithm that constructs, for each given finite state acceptor M, a hyper-minimized acceptor almost-equivalent to the original. More precisely, the algorithm will give one of possible solutions, since the hyper-minimized machine is not necessarily unique. Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA. Then:

(a) The given automaton M is made minimized, that is, if there exist unreachable states or pairs of equivalent states, they are removed. This can be done in the standard way [8], known for decades [9,11,13].

(b) For future use, in Step (c), reachability in the transition graph is analyzed. Let $Q = \{q_1, q_2, \ldots, q_n\}$, with $q_1 = q_1$. We first build a Boolean $n \times n$ matrix E, with $e_{i,j} = 1$ if and only if $\delta(q_i, \alpha) = q_j$, for some string $\alpha \in \Sigma^*$, $\alpha \neq \varepsilon$. That is, $e_{i,j} = 1$ if and only if q_j is reachable from q_i, ruling out trivial paths of length zero. For more details about computing reachability in graphs or Boolean transitive closure, the reader is referred to [1,3].

(c) Distinguish preamble states from kernel states. (Having executed Step (a), we have no unreachable states.) Therefore, we compute a Boolean vector K, in which $k_i = 1$ if and only if q_i is in the kernel. By Lemma 2.12, this only requires to determine, for each $q_i \in Q$, whether there exists a $q_j \in Q$ such that $e_{1,j} = 1$, $e_{j,j} = 1$, and $e_{j,i} = 1$.

(d) Now we need to determine which states are almost-equivalent. By Lemma 2.8, we do not have to keep this kind of information about each pair of states. Instead, we shall keep a partition of the state set into $Q = Q_1 \cup Q_2 \cup \ldots \cup Q_r$, for

some $r \geq 1$, such that each two states belonging to the same subset Q_i are *already known* to be almost-equivalent. The list $Q_1, Q_2, \ldots, Q_r$ can be implemented, *e.g.*, as an integer vector P, in which $p_i = j$ reflects the fact that $q_i \in Q_j$. (In a more sophisticated and more efficient implementation, one can utilize a data structure used to solve the so called Union-Find Problem [1].) Initially, we take $Q = \{q_1\} \cup \{q_2\} \cup \ldots \cup \{q_n\}$.

After that, we look for a state pair q_{i_1}, q_{i_2} not known to be almost-equivalent, but for which all respective direct descendants are almost-equivalent. That is, the algorithm searches for some $q_{i_1} \in Q_{i_1}$ and $q_{i_2} \in Q_{i_2}$, with $i_1 \neq i_2$, such that, for each $a \in \Sigma$, both $\delta(q_{i_1}, a)$ and $\delta(q_{i_2}, a)$ are contained in the same set Q_{j_a}. If such pair is found, the sets Q_{i_1} and Q_{i_2} are replaced in the list $Q_1, Q_2, \ldots, Q_r$ by a single new set $Q_{i_1} \cup Q_{i_2}$. This process is repeated until the moment when no new state pair can be found for set merging. Clearly, this is never repeated more that $n-1$ times.

Using Lemma 2.10, one can easily show that, in the course of computation, any two states belonging to the same set Q_i in the list $Q_1, Q_2, \ldots, Q_r$ are indeed almost-equivalent. That is, the algorithm does not produce a "wrong" partition of the states.

Moreover, the algorithm does not terminate "prematurely", with some states $q_A \sim q_B$ still belonging to some different sets in the list $Q_1, Q_2, \ldots, Q_r$. This follows from Theorem 3.1: if $q_A \sim q_B$, then, for each $\alpha \in \Sigma^*$ of length $|\alpha| \geq n^2$, we have $\delta(q_A, \alpha) \equiv \delta(q_B, \alpha)$, and hence also $\delta(q_A, \alpha) = \delta(q_B, \alpha)$, since equivalent states have already been removed in Step (a). Therefore, if we consider the two trees of all possible descendants of q_A and q_B in parallel, these two trees must contain some $q'_A \sim q'_B$ also belonging to different sets in $Q_1, Q_2, \ldots, Q_r$, but for which all their direct descendants are "already known" to be almost-equivalent. Therefore, the algorithm cannot stop at this moment, if nothing else, the pair q'_A, q'_B is detected for set merging.

(*e*) Finally, we can take out the useless states. Roughly speaking, all transitions ending in a useless state q_A are "redirected" from q_A to a substitute state q_B. The state q_A itself is removed. The detailed construction has been presented in the proof of Theorem 3.2. As already shown in this proof, the modified machine accepts a finite variation of the original language, if the following conditions are satisfied: $q_A \sim q_B$, q_A is in the preamble, and q_A is not reachable from q_B. (See also item (i) in this proof, remarks below this proof, as well as Lem. 2.13.)

Using these ideas, we can process the list $Q_1, Q_2, \ldots, Q_r$ (now dividing Q into classes of almost-equivalent states) in the following way.

- If, for some i, the set Q_i consists of kernel states only, or if it contains a single state, no states can be saved in Q_i. Recall that the kernel states are distinguished by the use of the Boolean vector K, computed in Step (c).
- If the set Q_i contains both preamble and kernel states, pick an arbitrary kernel state $q_B \in Q_i$ and redirect, for each preamble state $q_A \in Q_i$, all edges ending in q_A from q_A to q_B. The state q_A is removed. Clearly, q_A is not reachable from q_B or else q_A would also be in the kernel.

- Finally, if all states in Q_i are in the preamble, then pick an arbitrary state $q_B \in Q_i$ and redirect, for each state $q_A \in Q_i \setminus \{q_B\}$, all edges ending in q_A from q_A to q_B. The state q_A is removed. By Lemma 2.14, q_A is not reachable from q_B or else there would exist a kernel state $q_C \in Q_i$, which is a contradiction.

It should be pointed out that, by Theorem 3.4, the automaton produced by the above algorithm cannot be beaten by any other automaton, even after allowing any additional finite variations of the original language. It is also relatively easy to derive that the above algorithm runs in $O(n^3 \cdot m)$ time, where $n = \|Q\|$ and $m = \|\Sigma\|$. This gives an $O(n^3)$ algorithm for automata with binary inputs.

5. Hyper-minimized transducers

With minor modifications, most of the results presented in the sections above can be updated for finite state transducers, *i.e.*, for automata that do not recognize languages, but produce some outputs.

Equivalent transducers can be defined in the standard way, *i.e.*, they must produce the same output for each given input. Almost equivalent transducers must produce the same sequence of output symbols *after an initial "probation period" taking only a fixed amount of time*. The length of probation period is independent from the length of input. This can be formalized as follows.

Definition 5.1. A *deterministic finite state transducer* is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, \pi, q_I)$, where Q denotes a finite set of states, Σ a finite set of input symbols, Δ a finite set of output symbols, $\delta : Q \times \Sigma \to Q$ is a transition function, $\pi : Q \times \Sigma \to \Delta$ an output function, and $q_I \in Q$ is an initial state.

In one move, M reads an input symbol $a \in \Sigma$, changes its state from q to $q' = \delta(q, a)$, as usual, but produces also one symbol to the output, namely, $b = \pi(q, a) \in \Delta$.

The transition function δ is again extended to $\delta : Q \times \Sigma^* \to Q$ in the standard way, so that $\delta(q, \alpha_1\alpha_2) = \delta(\delta(q, \alpha_1), \alpha_2)$. Similarly, the output function π is extended to $\pi : Q \times \Sigma^* \to \Delta^*$, so that, for each $q \in Q$ and $\alpha_1, \alpha_2 \in \Sigma^*$, we have $\pi(q, \alpha_1\alpha_2) = \pi(q, \alpha_1) \cdot \pi(\delta(q, \alpha_1), \alpha_2)$.

Therefore, the machine M computes the function $f_M(\alpha) = \pi(q_I, \alpha)$.

Definition 5.2. In a deterministic transducer, two states q_A and q_B are *equivalent*, denoted by $q_A \equiv q_B$, if, for each $\alpha \in \Sigma^*$, we have $\pi(q_A, \alpha) = \pi(q_B, \alpha)$.

We say that two strings $\alpha = a_1 \dots a_n$ and $\beta = b_1 \dots b_n$ are *k-equal*, for some $k \geq 1$, which will be denoted by $\alpha \approx_k \beta$, if $a_k \dots a_n = b_k \dots b_n$. By definition, we take $a_k \dots a_n = \varepsilon$ for $n < k$, and hence all strings shorter than k are k-equal, if they are of equal length.

Two states q_A and q_B are *almost-equivalent*, denoted by $q_A \sim q_B$, if there exists a $k \geq 1$ such that, for each $\alpha \in \Sigma^*$, we have $\pi(q_A, \alpha) \approx_k \pi(q_B, \alpha)$.

The above definition is extended to pairs of states belonging to different machines, but using the same input and output alphabets, in the same way as in the

case of finite state acceptors. That is, having given $M_1 = (Q_1, \Sigma, \Delta, \delta_1, \pi_1, q_{\mathrm{I},1})$ and $M_2 = (Q_2, \Sigma, \Delta, \delta_2, \pi_2, q_{\mathrm{I},2})$, the states in $Q = Q_1 \cup Q_2$ are (almost-) equivalent, if they are (almost-) equivalent in the automaton $M = (Q, \Sigma, \Delta, \delta, \pi, q_{\mathrm{I},1})$, where $\delta(q,a) = \delta_1(q,a)$ if $q \in Q_1$, but $\delta(q,a) = \delta_2(q,a)$ if $q \in Q_2$, and $\pi(q,a) = \pi_1(q,a)$ if $q \in Q_1$, but $\pi(q,a) = \pi_2(q,a)$ if $q \in Q_2$.

Definition 5.3. Two deterministic finite state transducers M_1 and M_2 are *equivalent*, denoted by $M_1 \equiv M_2$, if $q_{\mathrm{I},1} \equiv q_{\mathrm{I},2}$. Similarly, M_1 and M_2 are *almost-equivalent*, denoted by $M_1 \sim M_2$, if $q_{\mathrm{I},1} \sim q_{\mathrm{I},2}$.

In other words, equivalent machines produce equal outputs, and hence $f_{M_1} = f_{M_2}$. On the other hand, almost-equivalent machines agree in outputs after an initial "probation period", that is, there exists a $k \geq 1$ such that, for each $\alpha \in \Sigma^*$, $f_{M_1}(\alpha) \approx_k f_{M_2}(\alpha)$, which can be denoted by $f_{M_1} \sim f_{M_2}$.

Minimal and hyper-minimized transducers can be defined in the standard way:

Definition 5.4. The finite state transducer M is said to be *minimized*, if no transducer M' with fewer states is equivalent to M. Similarly, M is *hyper-minimized*, if no M' with fewer states is almost-equivalent to M.

Clearly, $\beta_1 = \beta_2$ if and only if $\beta_1 \approx_1 \beta_2$, and hence equivalent transducers are also almost-equivalent. For the same reason, a hyper-minimized transducer M is also minimized, for the function f_M.

It should be obvious that many properties of finite state acceptors can be translated literally into a language of transducers, like, *e.g.*, Facts 2.5, 2.6, Lemma 2.8, Definition 2.11, and Lemmas 2.12–2.14. Some of them require only a slightly different formulation or a slightly modified argument. The next two statements are counterparts of Fact 2.7 and Lemma 2.9, respectively.

Fact 5.5. *$q_{\mathrm{A}} \equiv q_{\mathrm{B}}$ if and only if*

(*a*) *for each $a \in \Sigma$, $\delta(q_{\mathrm{A}}, a) \equiv \delta(q_{\mathrm{B}}, a)$, and*
(*b*) *for each $a \in \Sigma$, $\pi(q_{\mathrm{A}}, a) = \pi(q_{\mathrm{B}}, a)$.*

Lemma 5.6. *If $q_{\mathrm{A}} \sim q_{\mathrm{B}}$, then for each $\alpha \in \Sigma^*$, $\delta(q_{\mathrm{A}}, \alpha) \sim \delta(q_{\mathrm{B}}, \alpha)$.*

Proof. Suppose that, for some $\tilde{\alpha} \in \Sigma^*$, the state $q'_{\mathrm{A}} = \delta(q_{\mathrm{A}}, \tilde{\alpha})$ is *not* almost-equivalent to $q'_{\mathrm{B}} = \delta(q_{\mathrm{B}}, \tilde{\alpha})$. Using Definition 5.2, we get that, for each $k \geq 1$, there exists an $\alpha_k \in \Sigma^*$ such that $\pi(q'_{\mathrm{A}}, \alpha_k) \not\approx_k \pi(q'_{\mathrm{B}}, \alpha_k)$. Therefore, we have an infinite sequence of inputs $\alpha_1, \alpha_2, \ldots$ such that, for each $k = 1, 2, \ldots$, the output strings $\pi(q'_{\mathrm{A}}, \alpha_k)$ and $\pi(q'_{\mathrm{B}}, \alpha_k)$ differ in at least one symbol produced after the $(k-1)$-th computation step.

Now, by taking $\tilde{\alpha}\alpha_1, \tilde{\alpha}\alpha_2, \ldots$, we get an infinite sequence such that, for each $k = 1, 2, \ldots$, the outputs $\pi(q_{\mathrm{A}}, \tilde{\alpha}\alpha_k)$ and $\pi(q_{\mathrm{B}}, \tilde{\alpha}\alpha_k)$ also differ in a symbol produced after the $(k-1)$-th computation step. Therefore, for each $k \geq 1$, we have an input $\tilde{\alpha}\alpha_k \in \Sigma^*$ with $\pi(q_{\mathrm{A}}, \tilde{\alpha}\alpha_k) \not\approx_k \pi(q_{\mathrm{B}}, \tilde{\alpha}\alpha_k)$, which gives $q_{\mathrm{A}} \not\sim q_{\mathrm{B}}$.

Summing up, if $\delta(q_{\mathrm{A}}, \tilde{\alpha}) \not\sim \delta(q_{\mathrm{B}}, \tilde{\alpha})$, for some $\tilde{\alpha} \in \Sigma^*$, then $q_{\mathrm{A}} \not\sim q_{\mathrm{B}}$. □

In the same spirit, we can rework the proofs of Lemma 2.10 and Theorem 3.1. With this tools in our hands, we can easily obtain the following counterpart of Theorem 3.4.

Theorem 5.7. *A finite state transducer M is hyper-minimized if and only if, in M,*

(*a*) *there does not exist an unreachable state q_A; and*
(*b*) *there does not exist a pair of different but equivalent states $q_A \equiv q_B$; and*
(*c*) *there does not exist a pair of different but almost-equivalent states $q_A \sim q_B$, such that at least one of them is in the preamble.*

Without (c), this gives a characterization of minimized transducers, a counterpart of Fact 3.5. The argument combines the ideas presented in proofs of Theorems 3.2 and 3.3 with obvious differences. Here we take into account sequences of symbols produced to the output rather than the sequences of acceptance-values, but still using the same computation paths. (*Cf.* proofs of Lems. 5.6 and 2.9.)

We content ourselves with pointing out how transitions ending in a state q_A are redirected to another state q_B. The new transducer is $M' = (Q\backslash\{q_A\}, \Sigma, \Delta, \delta', \pi', q'_I)$, where

$$\begin{aligned}
\delta'(q,a) &= \begin{cases} \delta(q,a), & \text{if } \delta(q,a) \neq q_A, \\ q_B, & \text{if } \delta(q,a) = q_A, \end{cases} \\
\pi'(q,a) &= \pi(q,a), \\
q'_I &= \begin{cases} q_I, & \text{if } q_I \neq q_A, \\ q_B, & \text{if } q_I = q_A, \end{cases}
\end{aligned}$$

for each $q \in Q\backslash\{q_A\}$ and $a \in \Sigma$. (Compare with the construction presented in the proof of Th. 3.2.)

The new modified machine agrees in the sequence of output symbols after an initial period taking only a fixed amount of time, if the following conditions are satisfied: $q_A \sim q_B$, q_A is in the preamble, and q_A is not reachable from q_B. By Lemma 2.13, this can be used if the condition (c) of Theorem 5.7 does not hold, *i.e.*, if there does exist a pair of different but almost-equivalent states $q_A \sim q_B$, such that at least one of them is in the preamble.

As in the case of finite state acceptors, the minimized transducers are unique, up to isomorphism. It is not too complicated to find examples in which the given minimized n-state transducer M can be replaced by several different hyper-minimized machines, all of them using the same number of states, below n, but each computing a different function $f_{M'} \sim f_M$.

Theorem 5.8. *Let $M = (Q, \Sigma, \Delta, \delta, \pi, q_I)$ and $M' = (Q', \Sigma, \Delta, \delta', \pi', q'_I)$ be two almost-equivalent hyper-minimized finite state transducers. Then there exists a function $h\colon Q \to Q'$ with the following properties.*

- *If $q \in P$, then $q \sim h(q)$. If $q \in K$, then $q \equiv h(q)$.*
- *Between the preamble parts, the function is a bijection: 1 if $q \in P$, then $h(q) \in P'$, 2 for each $q_A, q_B \in P$, $q_A \neq q_B$ implies that $h(q_A) \neq h(q_B)$, and 3 for each $q' \in P'$, there exists a $q \in P$ such that $h(q) = q'$.*
- *Between the kernel parts, the function is a bijection: 1 if $q \in K$, then $h(q) \in K'$, 2 for each $q_A, q_B \in K$, $q_A \neq q_B$ implies that $h(q_A) \neq h(q_B)$, and 3 for each $q' \in K'$, there exists a $q \in K$ such that $h(q) = q'$.*

- *In the kernels, the function preserves the machine's behavior, except for the initial states:* $h(\delta(q,a)) = \delta'(h(q),a)$ *and* $\pi(q,a) = \pi'(h(q),a)$*, for each* $q \in K$ *and* $a \in \Sigma$*.*
- *In the preambles, the function preserves the machine's behavior, except for produced outputs and edges leaving the preamble: 1* $h(q_{\text{I}}) = q'_{\text{I}}$*, provided that* $q_{\text{I}} \in P$*, and 2* $h(\delta(q,a)) = \delta'(h(q),a)$*, for each* $q \in P$ *and* $a \in \Sigma$ *satisfying* $\delta(q,a) \in P$*.*

A little bit surprisingly, the algorithm for hyper-minimization of transducers is almost identical with the algorithm presented in Section 4. More precisely, Steps (b)–(e), converting a minimized automaton into an almost-equivalent hyper-minimized machine, are exactly the same. The existence of an output function π is simply ignored here, the set of final states F is not utilized.

The only difference is in Step (a), performing the classical minimization. First, unreachable states are removed in the standard way.

But, in order to remove equivalent states, a slightly different initialization is required. Recall that the standard minimizing algorithm [8], Section 4.4.3 partitions the states into blocks, so that all states in the same block are equivalent, and no pair of states from different blocks are equivalent. Initially, the partition consists of two blocks only, the first one contains all accepting states, the second one all remaining states. For finite state transducers, we use Fact 5.5 instead of Fact 2.7. Therefore, the initial partition may consist of more than two blocks. Initially, q_{A} and q_{B} are in the same block if and only if, for each $a \in \Sigma$, $\pi(q_{\text{A}},a) = \pi(q_{\text{B}},a)$. From this point forward, all subsequent iterations of the algorithm refining this initial partition proceed in the standard way, using only the transition function δ. The output function π or the set of final states F are no longer required.

6. Concluding remarks

We have devised a polynomial-time *hyper-minimization* algorithm that reduces a given deterministic finite state automaton into a hyper-minimized automaton, which may have fewer states than the classically minimized DFA. The language it accepts is only finitely different from the original. For each $m \leq n$, we can find an example of a minimized n-state automaton for which its hyper-minimized almost-equivalent variant uses exactly m states. In Theorem 3.4, we gave a characterization of hyper-minimized automata that may serve as a simple tool for testing hyper-minimality. (With small modifications, all constructions work for finite state transducers as well.)

The hyper-minimized automaton is not necessarily unique: in general, there may exist several different hyper-minimized machines, all of them using the same number of states, but each accepting a separate language finitely-different from the original one. In particular, acceptance values in the preamble may differ, transitions from the preamble to the kernel can change in restricted ways, and if the preamble is empty, the starting states might differ. However, there are

always large structural similarities among all these machines. In the unary case, the hyper-minimized automaton is always unique.

The properties of hyper-minimized automata led us to the finite-difference relation between two languages, introduced by Definition 2.3b, which is an equivalence relation. For example, if $L_1 \sim L_2$ and $L_2 \sim L_3$, then $L_1 \sim L_3$. Note also that the set of regular languages is closed under finite difference: if L_1 is regular and $L_1 \sim L_2$, then L_2 is regular. By this relation, the family of regular languages is partitioned into equivalence classes. As shown by Theorem 3.8, the common property of all languages within the same class is that the kernel parts of their minimized DFAs are isomorphic. From this point of view, the hyper-minimization algorithm finds, for each given automaton, an automaton that has the fewest states but accepts a language within the same class. Because of non-uniqueness, each class can have more than one such smallest member.

These smallest members constitute another language family, having at least one representative in each equivalence class under the finite difference relation. We can call it the family of *canonical regular languages*. A regular language L is canonical if the unique minimized automaton accepting L is already hyper-minimized. In other words, L is one of the smallest members within its equivalence class under the finite difference relation: each L' that can be accepted by a machine with fewer states must be infinitely different from L.

The standard complexity measure of a regular language L, used in the literature, is its *state complexity*, denoted by $s(L)$ and defined as the number of states of a minimized deterministic finite automaton for L. Taking into account the results presented above, it makes sense to consider also $s^*(L)$, an *asymptotic state complexity*, defined as the number of states of a hyper-minimized automaton for a language finitely-different from L. In a way, this measure factors out a finite initial subset and concentrates on how complex is the core of the regular language L. (For example, $s^*(L)=1$ for each finite or co-finite language L.)

We shall conclude this paper by presenting some open problems of theoretical and practical interest:

- What are the closure properties of canonical regular languages? We only know that this family forms a proper subset in the class of all regular languages and that it is closed under complement.
- Similarly, we know next to nothing about the asymptotic state complexity. For example, having given L_1 and L_2, what can be told about $s^*(L_1 \cup L_2)$?
- Nothing is known about hyper-minimized machines for other models, like nondeterministic, alternating, or two-way automata.
- Find a more efficient hyper-minimization algorithm. The algorithm described in Section 4 takes $O(n^3 \cdot m)$ time, where n is the number of states and m the size of the input alphabet. This is relatively far from the $O(n \cdot \log n)$ of standard minimization [9], and we strongly believe that this can be improved.
- Since there may exist several different hyper-minimized machines, it might be useful to design an algorithm that picks from among them the best

one according to some second-order criterion, *e.g.*, minimizing also the number of changed strings, or minimizing the size of the longest string with changed acceptance.

- In many practical applications, we may know *a priori* that a string shorter than some fixed value k will never be tested for membership. Devise an algorithm that constructs, for each given automaton M and each given $k \geq 0$, an automaton M' that does not differ in acceptance on inputs of length $|\alpha| \geq k$, minimizing the number of states in M'. Clearly, the size of M' monotonically decreases in k, varying within the limits given by the sizes of minimized and hyper-minimized automata.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley (1976).

[2] A. Bertoni, C. Mereghetti and G. Pighizzini, An optimal lower bound for nonregular languages. *Inform. Process. Lett.* **50** (1994) 289–292. (Corrigendum: *Inform. Process. Lett.* **52** (1994) 339).

[3] G. Brassard and P. Bratley, *Fundamentals of Algorithmics*. Prentice Hall (1996).

[4] M. Chrobak, Finite automata and unary languages. *Theoret. Comput. Sci.* **47** (1986) 149–158. (Corrigendum: *Theoret. Comput. Sci.* **302** (2003) 497–498).

[5] V. Geffert, (Non)determinism and the size of one-way finite automata, in *Proc. Descr. Compl. Formal Syst.* IFIP & Univ. Milano (2005) 23–37.

[6] V. Geffert, Magic numbers in the state hierarchy of finite automata, in *Proc. Math. Found. Comput. Sci.*, Springer-Verlag. *Lect. Notes Comput. Sci.* **4162** (2006) 412–423.

[7] V. Geffert, C. Mereghetti and G. Pighizzini, Converting two-way nondeterministic unary automata into simpler automata. *Theoret. Comput. Sci.* **295** (2003) 189–203.

[8] J. Hopcroft, R. Motwani and J. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley (2001).

[9] J.E. Hopcroft, An $n \log n$ algorithm for minimizing the states in a finite automaton, in *The Theory of Machines and Computations*, edited by Z. Kohave, pp. 189–196. Academic Press (1971).

[10] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley (1979).

[11] D.A. Huffman, The synthesis of sequential switching circuits. *J. Franklin Inst.* **257** (1954) 161–190 and 275–303.

[12] C. Mereghetti and G. Pighizzini, Optimal simulations between unary automata. *SIAM J. Comput.* **30** (2001) 1976–1992.

[13] E.F. Moore, Gedanken experiments on sequential machines, in *Automata Studies*, edited by C.E. Shannon and J. McCarthy, pp. 129–153. Princeton University Press (1956).

Communicated by J. Hromkovic.
Received July 12, 2007. Accepted November 21, 2007.