Minimizing NFA's and Regular Expressions

(Extended Conference Version)

Gregor Gramlich, Georg Schnitger *

Institut für Informatik
Johann Wolfgang Goethe–Universität Frankfurt
Robert-Mayer-Straße 11-15
60054 Frankfurt am Main, Germany
{gramlich,georg}@thi.informatik.uni-frankfurt.de
Fax: +49 - 69 - 798-28814

Published in STACS 2005
22nd International Symposium on Theoretical Aspects of Computer Science, pp. 399-411.
Lecture Notes in Computer Science 3404

Abstract. We show inapproximability results concerning minimization of nondeterministic finite automata (nfa's) as well as regular expressions relative to given nfa's, regular expressions or deterministic finite automata (dfa's). We show that it is impossible to efficiently minimize a given nfa or regular expression with n states, transitions, resp. symbols within the factor $o(n)$, unless P = PSPACE. Our inapproximability results for a given dfa with n states are based on cryptographic assumptions and we show that any efficient algorithm will have an approximation factor of at least $\frac{n}{\text{poly}(\log n)}$. Our setup also allows us to analyze the minimum consistent dfa problem.
Classification: Automata and Formal Languages, Computational Complexity, Approximability

1 Introduction

Among the most basic objects of formal language theory are regular languages and their acceptance devices, finite automata and regular expressions. Regular expressions describe lexical tokens for syntactic specifications, textual patterns in text manipulation systems and they are the basis of standard utilities such as scanner generators, editors or programming languages (perl, awk, php). Internally regular expressions are converted to (nondeterministic) finite automata and the succinctness of this representation crucially determines the running time of the applied algorithms.

Contrary to the problem of minimizing dfa's, which is efficiently possible, it is well known that nfa or regular expression minimization is computationally hard, namely PSPACE-complete [10]. Jiang and Ravikumar [7] show moreover that the minimization problem for nfa's or regular expressions remains PSPACE-complete, even when specifying the regular language by a dfa.

We consider the problem of *approximating* a minimal nfa or a minimal regular expression. In [3] it is shown that unary nfa's are hard to approximate and in particular efficient approximation algorithms require an approximation factor of at least $\frac{\sqrt{n}}{\ln n}$ for given nfa's or regular expressions of size n, provided $P \neq NP$. On the other hand, there are several approaches to nfa minimization [1, 4, 5, 9] without approximation guarantees or running in at least exponential time. This article explains why such guarantees cannot be expected for efficient algorithms.

We investigate the approximation problem in two scenarios. In the first scenario the language is specified by a dfa which makes proofs of inapproximability hard, since the input is not specified concisely and thus more time compared to concise inputs such as nfa's or regular expressions is available. Jiang and Ravikumar [7] ask to determine the approximation complexity of converting dfa's into nfa's, and in particular ask whether efficient approximation algorithms with a polynomial approximation factor exist. Corollary 1 shows that such an approximation is at least as

* partially supported by DFG project SCHN503/2-1

hard as factoring Blum integers and therefore efficient approximation algorithms with polynomial approximation factor are unlikely.

We show in Theorem 1 that efficient approximation algorithms determine regular expressions of length at least $\frac{k}{\mathrm{poly}(\log k)}$ for a given dfa of size k, even if optimal regular expressions of length $\mathrm{poly}(\log k)$ exist. We have to assume however that strong pseudo-random functions exist in non-uniform NC^1. The concept of a strong pseudo-random function is introduced by Razborov and Rudich [14]. Naor and Reingold [11] show that strong pseudo-random functions exist even in TC^0, provided factoring Blum integers requires time $2^{\Omega(n^\varepsilon)}$ (for some $\varepsilon > 0$).

We show similar results for approximating nfa's in Corollary 1, but now relative to the assumption that strong pseudo-random functions exist in non-uniform Logspace. We also apply our technique to the minimum consistent dfa problem [8, 12] in which a dfa of minimum size, consistent with a set of classified inputs, is to be determined.

Thus in the first scenario we follow the cryptographic approach of Kearns and Valiant [8] when analyzing the complexity of approximation, but work with pseudo-random functions instead of one-way functions. In the second scenario we assume that the language is specified by either an nfa or a regular expression. For the *unary* case we improve in Theorem 3 the approximation factor from $\frac{\sqrt{n}}{\ln n}$ [3] to $n^{1-\delta}$ for every $\delta > 0$, provided $P \neq NP$ and provided we require the approximation algorithm to determine a small equivalent nfa or regular expression, opposed to just determining the number of states.

Furthermore we show a PSPACE-completeness result for approximating the minimal size of *general* nfa's or regular expressions. Specifically Theorem 4 shows that it is impossible to efficiently minimize a given nfa or regular expression with n states, n transitions resp. n symbols within the factor $o(n)$, unless $P = \mathrm{PSPACE}$. The proof of Theorem 4 is based on the PSPACE-completeness of the "regular expression non-universality" problem.

We introduce strong pseudo-random functions in section 2 and investigate the complexity of approximating minimal regular expressions or nfa's, relative to a given dfa, in subsections 2.1 and 2.2. The minimum consistent dfa problem is considered in subsection 2.3. Finally the complexity of approximately minimizing unary resp. general nfa's or regular expressions, relative to a given nfa or regular expression, is determined in section 3.

2 Pseudo-Random Functions and Approximation

We consider the question of computing small equivalent nfa's or regular expressions for given dfa's. Inapproximability results seem to be hard to prove, since, intuitively, it takes large dfa's to specify hard inputs and consequently the allowed running time increases. Therefore we investigate the approximation complexity for minimum nfa's or regular expressions when given the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ and utilize the natural proof setup of Razborov and Rudich [14]. In particular, we utilize the concept of strong pseudo-random functions, but replace circuits by probabilistic Turing machines and require only a constant probability of separating pseudo-randomness from true randomness. Obviously strong pseudo-random functions exist in our setting, provided strong pseudo-random functions exist in the sense of Razborov and Rudich.

Definition 1. *Let $f_n = (f_n^s)_{s \in S}$ be a function ensemble with functions $f_n^s : \{0,1\}^n \to \{0,1\}$ for a seed $s \in S$ and let $(r_n^i)_{i \in \{1,\ldots,2^{2^n}\}}$ be the ensemble of all n-bit boolean functions. We call f_n a strong pseudo-random ensemble with parameter ε iff for any randomized algorithm A*

$$|\mathrm{prob}[A(f_n) = 1] - \mathrm{prob}[A(r_n) = 1]| < \frac{1}{3},$$

provided A runs in time $2^{O(n^\varepsilon)}$ and has access to f_n^s, resp. r_n^i, via a membership oracle. The probability is defined by the random choices of A and the uniform sampling of s from S, resp. the uniform sampling of i from $\{1, \ldots, 2^{2^n}\}$.

It is widely believed that there is some $\varepsilon > 0$, such that any algorithm running in time $2^{O(n^\varepsilon)}$ cannot factor Blum integers well on average. Naor and Reingold [11] construct TC^0 functions which are strong pseudo-random functions, provided factoring Blum integers requires time $2^{\Omega(n^\varepsilon)}$ for some ε.

Definition 2. *B_n is the set of all n-bit boolean functions. We define the compression $k_m : B_n \to B_m$ for $m < n$ by $(k_m(f))(x) = f(0^{n-m}x)$ for $x \in \{0,1\}^m$.*

We say, that a functional $G = (G_n)_n$ with $G_n : B_n \to \mathbb{N}$ separates a function class $\mathcal{C}$ from random functions with thresholds $t_1(\cdot)$ and $t_2(\cdot)$ iff $G_n(f) < t_1(n)$ holds for every function $f \in \mathcal{C} \cap B_n$, whereas $G_n(\rho) > t_2(n)$ for most functions in B_n, i.e., $|\{\rho \in B_n | G_n(\rho) \leq t_2(n)\}| = o(|B_n|)$ holds. Moreover we require that $G_m(k_m(f)) \leq t_1(n) \cdot \mathrm{poly}(n)$ for any function $f \in \mathcal{C} \cap B_n$ and any $m < n$.

It is not surprising that a functional G, which separates a function class $\mathcal{C}$ containing pseudo-random functions from random functions, cannot be efficiently approximated. We allow randomized approximation algorithms which may even underestimate the minimum.

Definition 3. *Let $|x|$ be the length of input x. We say that a randomized algorithm $App : X \to \mathbb{N}$ with approximation factor $\mu(|x|)$ for a minimization problem opt has overestimation error $\epsilon_+ = \sup_{x\in X} \mathrm{prob}[App(x) > \mu(|x|) \cdot opt(x)]$ and underestimation error $\epsilon_- = \sup_{x\in X} \mathrm{prob}[App(x) < opt(x)]$. The probabilities are defined by the random choices of App.*

We state a generic lemma for approximation algorithms on compressed inputs allowing us to replace oracle access by truth table presentation.

Lemma 1. *Assume that the functional G separates $\mathcal{C}$ from random functions with thresholds t_1, t_2 and suppose that $\mathcal{C}$ contains a strong pseudo-random ensemble with parameter ε.*

Let App be a randomized approximation algorithm that approximately determines $G_m(h_m)$, when given the truth table of a function $h_m \in B_m$. Then for all $l \geq 1$, if App runs in time $2^{O(m^l)}$ with errors $\epsilon_+ + \epsilon_- < \frac{2}{3}$, then App can only achieve an approximation factor $\mu_m \geq \frac{t_2(m)}{t_1(m^{l/\varepsilon})\mathrm{poly}(m^{l/\varepsilon})}$.

Proof. By assumption $\mathcal{C}$ contains strong pseudo-random functions with parameter ε. Let *App* be an algorithm which approximates $G_m(f_m)$ when given the truth table of f_m (with running time $2^{O(m^l)}$ for some $l \geq 1$, approximation factor $1 \leq \mu_m < \frac{t_2(m)}{t_1(m^{l/\varepsilon})\mathrm{poly}(m^{l/\varepsilon})}$ and errors $\epsilon_+ + \epsilon_- < \frac{2}{3}$). We construct an algorithm A which uses *App* to distinguish n-bit functions in $\mathcal{C}$ from n-bit random functions. We set $m = \lfloor n^{\varepsilon/l} \rfloor$.

A has oracle access to the input $h_n \in B_n$ and builds the truth table for the restriction $k_m(h_n)$. Then A runs *App* on $k_m(h_n)$ and accepts (i.e. $A(h_n) = 1$), if $App(k_m(h_n)) \leq t_2(m)$, and rejects (i.e. $A(h_n) = 0$) otherwise. So $|\mathrm{prob}[A(f_n) = 1] - \mathrm{prob}[A(r_n) = 1]| = |\mathrm{prob}[App(k_m(f_n)) \leq t_2(m)] - \mathrm{prob}[App(k_m(r_n)) \leq t_2(m)]|$ holds, where probabilities are taken over the probabilistic choices of *App* as well as the random sampling of seeds for f_n, respectively the uniform random sampling of functions $r_n \in B_n$.

G separates $\mathcal{C}$ from random functions and hence we have $G_m(k_m(f_n)) \leq t_1(n) \cdot \mathrm{poly}(n)$ for $f_n \subseteq \mathcal{C}$. Finally observe that $\mu_m \cdot t_1(n) \cdot \mathrm{poly}(n) < t_2(m)$ holds by assumption on μ_m.

$$\begin{aligned}\mathrm{prob}[App(k_m(f_n)) \leq t_2(m)] &\geq \mathrm{prob}[App(k_m(f_n)) \leq \mu_m \cdot t_1(n) \cdot \mathrm{poly}(n)]\\ &= 1 - \mathrm{prob}[App(k_m(f_n)) > \mu_m \cdot t_1(n) \cdot \mathrm{poly}(n)]\\ &\geq 1 - \mathrm{prob}[App(k_m(f_n)) > \mu_m \cdot G_m(k_m(f_n))]\end{aligned}$$

and $\mathrm{prob}[App(k_m(f_n)) \leq t_2(m)] \geq 1 - \epsilon_+$ follows. We utilize that a uniformly sampled function r_n from B_n leads to a uniformly sampled restriction from B_m.

$$\begin{aligned}\mathrm{prob}[App(k_m(r_n)) \leq t_2(m)] &\leq \mathrm{prob}[G_m(k_m(r_n)) \leq t_2(m)] + \epsilon_-\\ &= \frac{|\{\rho_m | G_m(\rho_m) \leq t_2(m)\}|}{|B_m|} + \epsilon_- = \epsilon_- + o(1).\end{aligned}$$

Thus $|\mathrm{prob}[App(k_m(f_n)) \leq t_2(m)] - \mathrm{prob}[App(k_m(r_n)) \leq t_2(m)]| \geq 1 - \epsilon_+ - \epsilon_- - o(1) > \frac{1}{3}$ holds for sufficiently large m. Since A runs in time $O(2^m) + 2^{O(m^l)} = 2^{O(n^\varepsilon)}$ this contradicts the assumption that $\mathcal{C}$ contains a strong pseudo-random ensemble with parameter ε. $\square$

In our first applications of Lemma 1, $G(f)$ will be the minimum length of regular expressions, respectively the minimum size of nfa's that accept, for some T, the complement of

$$L_T(f) = \{x^T | f(x) = 1\}.$$

2.1 Regular Expressions and Logarithmic Formula Depth

Definition 4. *A formula is a binary tree with $\wedge$ and $\vee$ gates as interior nodes; leaves are marked by labels from $\{x_1, \overline{x_1}, \ldots, x_i, \overline{x_i}, \ldots\}$. For a formula $\mathbf{f}$ let $\ell(\mathbf{f})$ be the length, i.e., the number of leaves of $\mathbf{f}$. The length $\ell(R)$ of a regular expression R is the number of symbols from the alphabet Σ appearing in R. The rpn-length of a regular expression R is the number of symbols from $\Sigma \cup \{+, \circ, ^*, \varepsilon, \emptyset\}$ appearing in R, when R is written in reverse Polish notation.*

Naor and Reingold [11] show that NC^1 contains a strong pseudo-random ensemble for some parameter $\varepsilon > 0$, provided factoring Blum integers is sufficiently hard. More precisely there is some constant c and a hard pseudo-random ensemble $\mathcal{C}_1$ with formula depth at most $c \cdot \log m$ for functions in $\mathcal{C}_1 \cap B_m$. Thus all functions in $\mathcal{C}_1 \cap B_m$ have formula length at most $T_1(m) = m^c$.

We define the functional $G^{(1)}$ by setting $G^{(1)}_m(f_m)$ to equal the minimum length of a regular expression for the complement of $L_{T_1}(f_m) = \{x^{T_1} | f_m(x) = 1\}$.

We associate regular expressions with formulae and show that the length of the regular expression is exponentially related to the depth of the formula.

Definition 5. *Let $\mathbf{f}$ be a formula for a function $f : \{0,1\}^m \to \{0,1\}$. We define the regular expression $R(\mathbf{f})$ recursively as follows:*

- *If $\mathbf{f} = x_i$, then $R(\mathbf{f}) := (0+1)^{i-1}\ 1\ (0+1)^{m-i}$.*
- *If $\mathbf{f} = \overline{x_i}$, then $R(\mathbf{f}) := (0+1)^{i-1}\ 0\ (0+1)^{m-i}$.*
- *If $\mathbf{f} = \mathbf{f}_1 \wedge \mathbf{f}_2$, then $R(\mathbf{f}) := R(\mathbf{f}_1) \circ R(\mathbf{f}_2)$.*
- *If $\mathbf{f} = \mathbf{f}_1 \vee \mathbf{f}_2$, then $R(\mathbf{f}) := R(\mathbf{f}_1) \circ (0+1)^{m \cdot \ell(\mathbf{f}_2)} + (0+1)^{m \cdot \ell(\mathbf{f}_1)} \circ R(\mathbf{f}_2)$.*

Lemma 2. *Let $W = \{w | \exists x \in \{0,1\}^m \wedge w \in \{x\}^*\}$ be the language of repeated inputs of length m.*

(a) $L(R(\mathbf{f})) \cap W = \{x^{\ell(\mathbf{f})} | f(x) = 1\} = L_{\ell(\mathbf{f})}(f)$.

(b) For a given formula $\mathbf{f}$ of depth k there is a regular expression $\overline{R_{\mathbf{f}}}$ which describes the complement of $L(R(\mathbf{f})) \cap W$. $\overline{R_{\mathbf{f}}}$ has length $O(4^k m)$ and can be constructed in time $\mathrm{poly}(4^k m)$.

(c) In particular, $\overline{L_{T_1}(f_m)}$ has regular expressions of length $t_1^{(1)} = O(m^{2c+1})$ for any $f_m \in \mathcal{C}_1 \cap B_m$.

Proof. (a) can be shown by an induction on the structure of formula $\mathbf{f}$.

(b) Moreover, such an induction shows that for a given formula $\mathbf{f}$ of depth k the regular expression $R(\mathbf{f})$ has length at most $2 \cdot 4^k m$ and can be constructed in time $\mathrm{poly}(4^k m)$. Observe that $\overline{L(R(\mathbf{f})) \cap W} = \overline{L(R(\mathbf{f}))} \cup \overline{W}$. We check whether the input does not consist of repetitions with the regular expression

$$\big((0+1)^*\ 1\ (0+1)^{m-1}\ 0\ (0+1)^*\big) + \big((0+1)^*\ 0\ (0+1)^{m-1}\ 1\ (0+1)^*\big)$$

and cover words of wrong length by $(0+1+\varepsilon)^{m \cdot \ell(\mathbf{f})-1} + (0+1)^{m \cdot \ell(\mathbf{f})+1}(0+1)^*$. We negate $\mathbf{f}$ with DeMorgan and observe that depth does not increase.

(c) Since all functions in $\mathcal{C}_1 \cap B_m$ have formula depth at most $c \cdot \log m$, we may assume that all these functions have formulae of depth exactly $c \cdot \log m$ and length exactly $T_1(m) = m^c$. Thus with part (a) $L_{T_1}(f_m)$ coincides with $L(R(\mathbf{f})) \cap W$ and, with part (b), $\overline{L_{T_1}(f_m)}$ has regular expressions of length $O(4^{c \log m} m) = O(m^{2c+1})$. $\square$

Thus we know that (strong pseudo-random) functions of formula depth at most $c \cdot \log m$ have short regular expressions of length at most $t_1(m) = \mathrm{poly}(m)$, whereas we show next that most m-bit functions have only regular expressions of length at least $\Omega(2^m)$.

Lemma 3. *The number of languages described by regular expressions of length at most $t_2^{(1)}(m) = \frac{2^m}{40}$ is bounded by $\sqrt{2^{2^m}} = o(|B_m|)$.*

Proof. A regular expression of length at most t has rpn-length at most $6t$ [5]. At any position in the regular expression in reverse Polish notation there may be one of the seven distinct symbols $0, 1, +, \circ, ^*, \varepsilon, \emptyset$. Thus we can have at most $\sum_{j \le 6t} 7^j \le 7^{7t} \le 2^{20t}$ distinct regular expressions of rpn-length at most $6t$. The claim follows, since $2^{20 t_2(m)} = 2^{20 \frac{2^m}{40}} = 2^{2^{m-1}}$. $\square$

$G^{(1)}_m(k_m(f_n)) \leq t^{(1)}_1(n)$ holds for functions $f_n \in \mathcal{C}_1 \cap B_n$, because a formula for f_n can be transformed into a formula for $k_m(f_n)$ of same depth. Hence as a consequence of Lemma 2 and Lemma 3, $G^{(1)}$ separates $\mathcal{C}_1$ from random functions with thresholds $t^{(1)}_1(m) = O(m^{2c+1})$ and $t^{(1)}_2(m) = \frac{2^m}{40}$.

Thus we may apply the generic Lemma 1 and obtain that efficient algorithms approximating the length of a shortest regular expression for $\overline{L_{T_1}(f)}$ do not exist. However we have to specify the input not by a truth table but by a dfa.

Proposition 1. *Let $f \in B_m$ and let T be some function of m, then there is a dfa $D_T(f)$ with $O(2^m \cdot T)$ states that accepts $\overline{L_T(f)}$.*

Proof. The dfa $D_T(f)$ consists of a binary tree of depth m rooted at the initial state. A leaf that corresponds to a word x with $f(x) = 0$ gets a self loop, a leaf that corresponds to a word x with $f(x) = 1$ is starting point of a path of length $(T-1)m$ that can only be followed by inputs with $T-1$ repetitions of x. Each such path leads to a rejecting state and any wrong letter on this path, resp. any word longer than $(T-1)m$ (measured on the path only) leads to an accepting trap state. Each state is accepting, except for those already described as rejecting. The dfa $D_T(f)$ has $O(2^m \cdot T)$ states. $\square$

Thus our first main result is now an immediate consequence of Lemma 1.

Theorem 1. *Suppose that strong pseudo-random functions with parameter ε and formula depth bounded by $c \cdot \log m$ exist for some c.*

Let App be a randomized approximation algorithm that approximately determines the length of a shortest equivalent regular expression, when given a dfa with k states. Then for all $l \geq 1$, if App runs in time $2^{O\left((\log k)^l\right)}$ with $\epsilon_+ + \epsilon_- < \frac{2}{3}$, then App can only achieve an approximation factor $\mu \geq \frac{k}{\mathrm{poly}((\log k)^{l/\varepsilon})}$.

The argument shows that there are always dfa's with optimal nfa's of size $\mathrm{poly}(\log k)$, such that an "efficient" approximation algorithm can only determine nfa's of size $\frac{k}{\mathrm{poly}(\log k)}$. Thus the original question of Jiang and Ravikumar [7] phrased for regular expressions instead of nfa's, namely whether it is possible to approximate within a polynomial, has a negative answer modulo cryptographic assumptions.

2.2 NFA's and Two-Way Automata of Polynomial Size

Here we use the functionals $G^{(2)}$ and $G^{(3)}$ defined by $G^{(2)}_m(f_m)$, resp. $G^{(3)}_m(f_m)$, to equal the minimum number of states, resp. transitions, of an nfa recognizing $\overline{L_{T_1}(f_m)}$. We choose T_1 as defined in the previous section and define $t^{(2)}_1 = t^{(1)}_1$, $t^{(3)}_1 = \left(t^{(1)}_1\right)^2$. We observe that the number of states of a minimum nfa is not larger than the length ℓ of an equivalent regular expression and the number of transitions is at most quadratic in ℓ. Thus all functions in $\mathcal{C}_1$ have nfa's of "size" at most $t^{(2)}_1$, resp. $t^{(3)}_1$. Moreover all but a negligible fraction of languages require nfa's with at least $t^{(2)}_2(m) = 2^{\frac{m}{2}-1}$ states, resp. $t^{(3)}_2(m) = \frac{2^m}{20m}$ transitions.

Lemma 4. *(a) The number of languages accepted by nfa's with at most $t^{(2)}_2(m) = 2^{\frac{m}{2}-1}$ states is bounded by $\sqrt{2^{m+2^m}} = o(|B_m|)$.*

(b) The number of languages accepted by nfa's with at most $t^{(3)}_2(m) = \frac{2^m}{20m}$ transitions is bounded by $\sqrt{2^{2^m}} = o(|B_m|)$.

Proof. (a) Let $N(k)$ be the number of distinct languages accepted by nfa's with at most k states over a two-letter alphabet. Then $N(k) \leq 2k \cdot 2^{2 \cdot k^2}$ [2] and hence $N(t^{(2)}_2(m)) \leq \frac{2}{2} \cdot 2^{\frac{m}{2}} \cdot 2^{2 \cdot \left(2^{\frac{m}{2}}/2\right)^2} = 2^{\frac{m}{2}} \cdot 2^{\frac{2}{4} \cdot \left(2^{(m/2)}\right)^2} = 2^{\frac{m}{2}} \cdot 2^{\frac{2^m}{2}} = \sqrt{2^{m+2^m}}$.

(b) We show that there are at most $M(k) = k^{10k}$ languages accepted by nfa's with at most k transitions over a two-letter alphabet. This establishes the claim, if we set $t^{(3)}_2(m) = \frac{2^m}{20m}$, since $M(t^{(3)}_2(m)) = \left(\frac{2^m}{20m}\right)^{10\frac{2^m}{20m}} \leq 2^{10m \cdot \frac{2^m}{20m}} = \sqrt{2^{2^m}}$.

For any nfa N with s states and k transitions there is an equivalent nfa N' with $s+1$ states, at most $2k$ transitions and exactly one final state. Just add a final state f, make every other state non-final and for every transition in N that leads to a final state in N, add a transition to f and keep every other transition.

There are at most $\binom{(s+1)^2}{2k}^2 \cdot s^2 \leq s^{8k+2}$ distinct languages over $\{0,1\}$ accepted by nfa's with s states and k transitions, since this is an upper bound for the number of possibilities to place $2k$ transitions for each letter of the alphabet $\{0,1\}$ and the number of choices for the initial and the final state.

We can assume that the number of states is bounded by the number of transitions and hence we have at most $k^{8k+2} \leq k^{10k}$ distinct languages. □

We apply Lemma 1 again and obtain:

Corollary 1. *Suppose that strong pseudo-random functions with parameter ε and formula depth bounded by $c \cdot \log m$ exist for some c.*

Let App be a randomized approximation algorithm that approximately determines the number of states (resp. number of transitions) of a minimum equivalent nfa, when given a dfa with k states. Then for all $l \geq 1$, if App runs in time $2^{O\left((\log k)^l\right)}$ with $\epsilon_+ + \epsilon_- < \frac{2}{3}$, then App can only achieve an approximation factor $\mu \geq \frac{\sqrt{k}}{\mathrm{poly}((\log k)^{l/\varepsilon})}$ (resp. $\mu \geq \frac{k}{\mathrm{poly}((\log k)^{l/\varepsilon})}$).

We finally mention that the assumption of strong pseudo-random functions with small formula depth can be replaced by the weaker assumption of strong pseudo-random functions with two-way dfa's of polynomial size. (Observe that two-way dfa's of polynomial size have the power of non-uniform Logspace, which is at least as powerful as non-uniform NC^1.) We show that two-way dfa's can be simulated efficiently by nfa's after repeating the input suitably often.

Lemma 5. *Let $m, k \in \mathbb{N}$ and let A_m be a two-way deterministic finite automaton with at most m^k states. Then there is a polynomial $T(m)$ and an nfa N_m with $O(T(m))$ states that accepts the complement of*

$$L_T(A_m) := \{x^{T(m)} \mid x \in \{0,1\}^m \wedge A_m \text{ accepts } x\}.$$

Proof. Obviously A_m runs for at most $T(m) = m \cdot m^k$ steps, since no cell can be visited twice in the same state. As shown in [13], A_m on input $x \in \{0,1\}^m$ can be simulated by a dfa D_m with $T(m)$ states working on input $x^{T(m)}$. The nfa N_m decides nondeterministically to run D_m (with final and non-final states interchanged) or to check whether the input is syntactically incorrect, i.e., verifying inequality or incorrect length. N_m has $t_1(m) = \mathrm{poly}(m)$ states, resp. transitions. □

When applying Lemma 1, we have to first redefine the number of repetitions to make sure that a class $\mathcal{C}_2$ of pseudo-random functions can be recognized by two-way dfa's of size m^k. We therefore set $T_2(m) = m^{k+1}$ and are guaranteed to find an equivalent nfa recognizing $\overline{L_{T_2}(f_m)}$ (for $f_m \in \mathcal{C}_2 \cap B_m$) with $t_1^{(2)}(m) = t_1^{(3)}(m) = O(T_2(m))$ states, resp. transitions.

2.3 The Minimum Consistent DFA Problem

In the *minimum consistent dfa* problem, sets $POS, NEG \subseteq \{0,1\}^*$ with $POS \cap NEG = \emptyset$ are given. The goal is to determine the minimum size of a dfa D such that $POS \subseteq L(D)$ and $NEG \cap L(D) = \emptyset$.

We again work with $T_2(m)$ repetitions and define $G_m^{(4)}(f_m)$ as the minimum size of a dfa accepting $POS = \{x^{T_2} | f_m(x) = 1\}$ and rejecting $NEG = \{x^{T_2} | f_m(x) = 0\}$. Observe that for any function $f_m \in \mathcal{C}_2 \cap B_m$ we have $G_m^{(4)}(f_m) \leq t_1^{(4)}(m) := m^{k+1}$, since any two-way dfa with m^k states can be simulated by a dfa with m^{k+1} states, if the input $x \in \{0,1\}^m$ is repeated $T_2(m) = m^{k+1}$ times. (See the proof of Lemma 5).

Lemma 6. $G_m^{(4)}(f_m) \leq t_2^{(4)}(m) = \frac{2^m}{6m}$ *holds for at most* $\sqrt{2^{2^m}} = o(|B_m|)$ *functions in* B_m.

Proof. Let $K(s)$ be the number of distinct languages accepted by dfa's with at most s states over a two-letter alphabet. Then $K(s) \leq s^{3s}$ [2] and hence $K(t_2^{(4)}(m)) \leq \left(\frac{2^m}{6m}\right)^{3\frac{2^m}{6m}} \leq 2^{3m\frac{2^m}{6m}} = \sqrt{2^{2^m}}$. The claim holds, since different functions f_m have different consistent dfa's. □

Thus $G_m^{(4)}$ separates $\mathcal{C}_2$ from random functions with thresholds $t_1^{(4)}, t_2^{(4)}$ and we obtain the following Theorem.

Theorem 2. *Suppose that strong pseudo-random functions with parameter ε and two-way dfa's with at most m^k states exist for some k.*

Let App be a randomized approximation algorithm that approximately determines the number of states of a minimum consistent dfa. For input length $N = \sum_{x \in POS \cup NEG} |x|$ and for all $l \geq 1$, if App runs in time $2^{O\left((\log N)^l\right)}$ with $\epsilon_+ + \epsilon_- < \frac{2}{3}$, then App can only achieve an approximation factor $\mu \geq \frac{N}{\text{poly}((\log N)^{l/\varepsilon})}$.

Efficient approximation algorithms determine, for $d \leq N$ examples, consistent dfa's of size $\frac{N}{\text{poly}(\log N)}$, whereas optimal dfa's have size $opt = \text{poly}(\log N)$. Thus upper bounds have as many as $2^{opt^{\frac{1}{l}}} \cdot d^\beta$ states, where $\beta < 1$ and l is sufficiently large. This result is stronger than the result of at least $opt^\alpha \cdot d^\beta$ due to Kearns and Valiant [8]. The stronger result is a consequence of our use of pseudo-random functions instead of one-way functions. (See also Naor and Reingold [11].)

3 Minimizing NFA's or Regular Expressions

We now assume that the language is specified concisely, i.e., as an nfa or a regular expression and prove in this scenario strong inapproximability results. We begin by investigating unary languages, i.e., languages over a one-letter alphabet, and show that no significant approximation is achievable, provided $P \neq NP$. This statement holds for size interpreted as number of states, transitions, resp. symbols.

Efficient approximations for state minimization within the factor $\frac{\sqrt{m}}{\ln m}$ are known *not* to exist, if $P \neq NP$ [3]. This result remains true for the number of transitions (resp. number of symbols in regular expressions), since the nfa (resp. regular expression) built by the transformation in the proof [3, 15] has as many states as transitions (resp. symbols), and the number of states is a lower bound for the number of transitions of a minimal equivalent nfa (resp. symbols of a minimal equivalent regular expression). We can improve the inapproximability result, if we require the *construction* of a small nfa or regular expression.

Theorem 3. *Let A be an arbitrary unary nfa or regular expression of size m. Let opt be the size of a minimal equivalent nfa, resp. regular expression. For any $\delta > 0$, if $P \neq NP$, then no efficient algorithm can determine an nfa or regular expression A' equivalent to A with size at most $opt \cdot m^{1-\delta}$.*

Proof. Let A be an nfa (regular expression) constructed in the NP-completeness proof [3, 15]. A has the property that either $opt = 1$ or $opt > \frac{\sqrt{m}}{\ln m}$ and it is NP-complete to distinguish the two cases.

Suppose that there is a constant $\delta > 0$ and an efficient algorithm M that computes an nfa (regular expression) $M(A)$ equivalent to A with $\text{size}(M(A)) \leq opt \cdot \text{size}(A)^{1-\delta}$. If we apply M on its output again, then $\text{size}(M(M(A))) \leq opt \cdot \text{size}(M(A))^{1-\delta} \leq opt^2 \cdot \text{size}(A)^{(1-\delta)^2}$. If we repeat this process k times, then $\text{size}(M^k(A)) \leq opt^k \cdot \text{size}(A)^{(1-\delta)^k}$. So for $k \geq \frac{-2}{\log(1-\delta)}$, we have $\text{size}(M^k(A)) \leq opt^k \cdot \text{size}(A)^{\frac{1}{4}}$, hence for m large enough, $\text{size}(M^k(A)) \leq \frac{\sqrt{m}}{\ln m}$ if $opt = 1$ and $\text{size}(M^k(A)) \geq opt > \frac{\sqrt{m}}{\ln m}$ otherwise. □

Our negative results for *general* alphabets are based on the well known proof [10] of the PSPACE-completeness of "regular expression non-universality": Given a regular expression R, is $L(R) \neq \Sigma^*$? The PSPACE-completeness of regular expression non-universality implies the PSPACE-completeness of the exact minimization of nfa's and regular expressions.

The proof of [10] shows, that for an arbitrary language $L \in$ PSPACE there is a (generic) polynomial time transformation T such that $w \in L \Leftrightarrow L(T(w)) \neq \Sigma^*$, where $L(T(w))$ is the language described by the nfa, resp. regular expression $T(w)$. We restrict ourselves to languages $L \in \mathcal{L}$ where $\mathcal{L}$ is the class of languages that can be accepted by deterministic in-place Turing machines[1]. Our inapproximability result utilizes the following observation.

[1] $\mathcal{L}$ coincides with DSPACE($O(n)$), but considering only Turing machines that work in-place simplifies the proof.

Lemma 7. *For any given language $L \in \mathcal{L}$ there is a deterministic in-place Turing machine M_L recognizing L with a single accepting state. M_L runs for at least 2^n steps on every input $w \in L$ of length n.*

Proof. Let M be some deterministic in-place Turing machine which accepts L and has only one accepting state q_f. We construct a Turing machine M_L that has all the states and transitions M has. However, whenever M_L enters q_f, it counts in binary from 0^n to 1^n, changes to a new state q'_f, when reaching 1^n, and stops. q'_f is the only state in which M_L accepts and q'_f causes M_L to stop. □

Assume that M is a Turing machine with the properties stated in Lemma 7 which recognizes the PSPACE-complete language $L(M)$. (A padding argument shows that $\mathcal{L}$ contains PSPACE-complete languages.) We reduce the word problem for $L(M)$ to the minimization problem for nfa's. In particular for an input w of M, we construct an nfa A_w, which accepts exactly all words which are *not* concatenations of consecutive legal configurations starting from configuration q_0w leading to the unique accepting state. The exact description of the construction of A_w is omitted. It shows that A_w has $m = O(|w|)$ states.

If M rejects w, then $L(A_w)$ coincides with Σ^*. However, if M accepts w, then the configuration sequence x corresponding to the accepting computation is rejected by A_w and it is the only rejected word.

We show that $\Sigma^* \setminus \{x\}$ requires nfa's with at least $|w|$ states. Any accepting computation has length at least $2^{|w|}$, since M is a Turing-Machine as described in Lemma 7. Every dfa which excludes a single word of length at least $2^{|w|}$ needs at least $2^{|w|}$ states, thus every equivalent nfa needs at least $|w|$ states. Hence, if $L(A_w) = \Sigma^* \setminus \{x\}$ for some x with $|x| \geq 2^{|w|}$, then every nfa which accepts $L(A_w)$ needs at least $|w|$ states.

Thus, if $w \notin L(M)$, then $L(A_w)$ can be recognized by an nfa with one state, whereas for $w \in L(M)$, nfa's with at least $|w|$ states are required. Since A_w has $m = O(|w|)$ states, we have found the desired gap.

The inapproximability result for the number of transitions of nfa's and the number of symbols in regular expressions follows along the same lines.

Theorem 4. *Unless $P =$ PSPACE, it is impossible to efficiently approximate the size of a minimal nfa or regular expression describing $L(A)$ within an approximation factor of $o(m)$ when given an nfa or a regular expression A with m states, transitions or symbols respectively.*

Standard encoding arguments show that this PSPACE-completeness result is true for regular expressions or nfa's over any alphabet Σ with $|\Sigma| \geq 2$.

4 Conclusions and an Overview

We have been able to verify inapproximability of nfa's or regular expressions either for given nfa's or regular expressions (utilizing $P \neq NP$, resp. $P \neq$ PSPACE) or for given dfa's (assuming the existence of strong pseudo-random functions in NC^1, resp. Logspace).

The most notably open problem is a negative result for given dfa's utilizing only $P \neq NP$. Furthermore, what is the approximation complexity, when specifying a regular language $L \subseteq \{0,1\}^n$ by a truth table? Below we list our results and additionally mention nfa minimization for a given unary dfa as a third important open problem.

NFA MINIMIZATION	
INSTANCE:	An nfa N with k states over a binary alphabet.
SOLUTION:	The size of a smallest nfa equivalent with N.
MEASURE:	Number of transitions or number of states.
BAD NEWS:	Not approximable within $o(k)$.
ASSUMPTION:	$P \neq$ PSPACE.
REFERENCE:	Theorem 4

Regular Expression Minimization

Instance: A regular expression R with k symbols over a binary alphabet.
Solution: The size of a smallest regular expression equivalent with R.
Measure: Number of symbols.
Bad News: Not approximable within $o(k)$.
Assumption: $P \neq$ PSPACE.
Reference: Theorem 4

The same is true for nfa $\rightarrow$ regular expression minimization and vice versa.

Unary NFA Minimization

Instance: An nfa N with k states over a unary alphabet.
Solution: The size of a smallest nfa equivalent with N.
Measure: Number of transitions or number of states.
Bad News: Not approximable within $\frac{\sqrt{k}}{\ln k}$.
Assumption: $P \neq NP$.
Reference: [3]

Constructive Unary NFA Minimization

Instance: An nfa N with k states over a unary alphabet.
Solution: A smallest nfa equivalent with N.
Measure: Number of transitions or number of states.
Bad News: Not approximable within $k^{1-\delta}$ for any δ.
Assumption: $P \neq NP$.
Reference: Theorem 3

DFA $\rightarrow$ NFA Minimization (States)

Instance: A dfa D with k states over a binary alphabet.
Solution: The size of a smallest nfa equivalent with D.
Measure: Number of states.
Bad News: Not approximable within $\frac{\sqrt{k}}{\mathrm{poly}(\log k)}$.
Assumption: Strong pseudo-random functions in Logspace.
Reference: Corollary 1

DFA $\rightarrow$ NFA Minimization (Transitions)

Instance: A dfa D with k states over a binary alphabet.
Solution: The size of a smallest nfa equivalent with D.
Measure: Number of transitions.
Bad News: Not approximable within $\frac{k}{\mathrm{poly}(\log k)}$.
Assumption: Strong pseudo-random functions in Logspace.
Reference: Corollary 1

Unary DFA $\rightarrow$ NFA Minimization

Instance: A dfa D with k states over a unary alphabet.
Solution: The size of a smallest nfa equivalent with D.
Measure: Number of states or transitions.
Bad News: Optimal solution cannot be determined efficiently.
Assumption: $NP \not\subseteq DTIME(n^{O(\log n)})$
Good News: Cyclic case can be approximated within $1 + \ln k$.
Reference: [6], [3]

DFA $\rightarrow$ Regular Expression Minimization	
Instance:	A dfa D with k states over a binary alphabet.
Solution:	The size of a smallest regular expression equivalent with D.
Measure:	Number of symbols.
Bad News:	Not approximable within $\frac{k}{\text{poly}(\log k)}$.
Assumption:	Strong pseudo-random functions in NC^1.
Reference:	Theorem 1

Minimum Consistent DFA	
Instance:	Two finite sets P, N of binary strings.
Solution:	The minimal size of a dfa accepting all strings in P and rejecting all strings in N.
Measure:	Number of states in the automaton.
Bad News:	Not approximable within $\frac{\lvert P\rvert+\lvert N\rvert}{\text{poly}(\log(\lvert P\rvert+\lvert N\rvert))}$.
Assumption:	Strong pseudo-random functions in Logspace.
Reference:	Theorem 2

References

1. Champarnaud, J.-M., Coulon, F.: NFA Reduction Algorithms by Means of Regular Inequalities, Developments in Language Theory, Springer, LNCS 2710, pp. 194-205.
2. Domaratzki, M., Kisman, D. Shallit, J.: On the Number of Distinct Languages Accepted by Finite Automata with n States, Journal of Automata, Languages and Combinatorics, 7(4), 2002.
3. Gramlich, G.: Probabilistic and Nondeterministic Unary Automata, Proc. of Math. Foundations of Computer Science, Springer, LNCS 2747, 2003, pp. 460-469.
4. Ilie, L., Navarro, G., Yu, S.: On NFA reductions, in J. Karhumaki, H. Maurer, G. Paun, G. Rozenberg, eds., Theory is Forever (Salomaa Festschrift), Springer, LNCS 3113, 2004, pp. 112-124.
5. Ilie, L., Yu, S.: Follow automata, Informat. & Computation 186, 2003, pp. 140-162.
6. Jiang, T., McDowell, E., Ravikumar, B.: The structure and complexity of minimal NFA's over a unary alphabet, Int. J. Found. of Comp. Sci., 2, 1991, pp. 163-182.
7. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard, SIAM Journal on Computing, 22(1), 1993, pp. 1117-1141.
8. Kearns, M., Valiant, L. G.: Cryptographic Limitations on Learning Boolean Formulae and Finite Automata, Journal of the ACM, 41(1), 1994, pp. 67-95.
9. Matz, O., Potthoff, A.: Computing small nondeterministic finite automata, Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Dpt. of CS., Univ. of Aarhus 1995, pp. 74-88.
10. Meyer, A. R., Stockmeyer, L. J.: The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space, Proc. 13th Ann. IEEE Symp. on Switching and Automata Theory, 1972, pp. 125-129.
11. Naor, M., Reingold, O.: Number-Theoretic constructions of efficient pseudo-random functions, Journal of the ACM, 51(2), 2004, pp. 231-262.
12. Pitt, L., Warmuth, M. K.: The Minimum Consistent DFA Problem Cannot be Approximated within any Polynomial, Journ. of the ACM, 1993, 40, pp. 95-142.
13. Pitt, L., Warmuth, M. K.: Prediction-Preserving Reducibility, Journal of Computer and System Science, 41(3), 1990, pp. 430-467.
14. Razborov, A. A., Rudich, S.: Natural Proofs, Journal of Computer and Systems Sciences, 55, 1997, pp. 24-35.
15. Stockmeyer, L., Meyer, A.: Word Problems Requiring Exponential Time, Proc. of the 5th Annual ACM Symposium on Theory of Computing, 1973, pp. 1-9.

A Appendix

Proof (of Lemma 2). Obviously $L(R(\mathbf{f})) \subseteq \{0,1\}^{m\cdot\ell(\mathbf{f})}$. We prove the lemma by induction on the depth of $\mathbf{f}$. If $\mathbf{f} = x_i$, then $R(\mathbf{f}) = (0+1)^{i-1}\, 1\, (0+1)^{m-i}$ and thus

$$L(R(\mathbf{f})) \cap W = \{x | x \in \{0,1\}^m \wedge x_i = 1\} = \{x | f(x) = 1\}.$$

The case $\mathbf{f} = \overline{x_i}$ follows analogously. If $\mathbf{f} = \mathbf{f}_1 \wedge \mathbf{f}_2$, then $R(\mathbf{f}) = R(\mathbf{f}_1) \circ R(\mathbf{f}_2)$ and thus

$$\begin{aligned} L(R(\mathbf{f})) \cap W &= (L(R(\mathbf{f}_1)) \circ L(R(\mathbf{f}_2))) \cap W \\ &= ((L(R(\mathbf{f}_1)) \cap W) \circ (L(R(\mathbf{f}_2)) \cap W)) \cap W \\ &= \Big(\{x^{\ell(\mathbf{f}_1)} | f_1(x) = 1\} \circ \{x^{\ell(\mathbf{f}_2)} | f_2(x) = 1\}\Big) \cap W \\ &= \{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} | f_1(x) = 1 \wedge f_2(x) = 1\} \\ &= \{x^{\ell(\mathbf{f})} | f(x) = 1\}. \end{aligned}$$

If $\mathbf{f} = \mathbf{f}_1 \vee \mathbf{f}_2$, then $R(\mathbf{f}) = R(\mathbf{f}_1) \circ (0+1)^{m\cdot\ell(\mathbf{f}_2)} + (0+1)^{m\cdot\ell(\mathbf{f}_1)} \circ R(\mathbf{f}_2)$ and thus

$$\begin{aligned} L(R(\mathbf{f})) \cap W &= (L(R(\mathbf{f}_1)) \circ \{0,1\}^{m\cdot\ell(\mathbf{f}_2)} \;\cup\; \{0,1\}^{m\cdot\ell(\mathbf{f}_1)} \circ L(R(\mathbf{f}_2))) \cap W \\ &= ((L(R(\mathbf{f}_1)) \circ \{0,1\}^{m\cdot\ell(\mathbf{f}_2)}) \cap W) \;\cup\; ((\{0,1\}^{m\cdot\ell(\mathbf{f}_1)} \circ L(R(\mathbf{f}_2))) \cap W) \\ &= \{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} | f_1(x) = 1\} \;\cup\; \{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} | f_2(x) = 1\} \\ &= \{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} | f_1(x) = 1 \vee f_2(x) = 1\} \\ &= \{x^{\ell(\mathbf{f})} | f(x) = 1\}. \end{aligned}$$

Let $\ell(k)$ be the length of $R(\mathbf{f})$ for a formula $\mathbf{f}$ with depth k. We show recursively that $\ell(k) \leq 2 \cdot 4^k m$. For $k = 0$ we have $\ell(0) \leq 2m = 2 \cdot 4^k m$. For formulae $\mathbf{f}_1$ and $\mathbf{f}_2$ of depth at most k the regular expression $R(\mathbf{f}_1 \wedge \mathbf{f}_2)$ has length at most $2\ell(k) \leq 2 \cdot 2 \cdot 4^k m = 4^{k+1} m$, and the regular expression $R(\mathbf{f}_1 \vee \mathbf{f}_2)$ has length at most $2\ell(k) + 2m \leq 2 \cdot 2 \cdot 4^k m + 2m \leq 2 \cdot 4^{k+1} m$.

We are done, since the explicit recursive construction in Definition 5 can be carried out in polynomial time. □

Proof (of Theorem 4). We give a complete proof of Theorem 4 and refer to the proof sketch in section 3 for motivating comments.

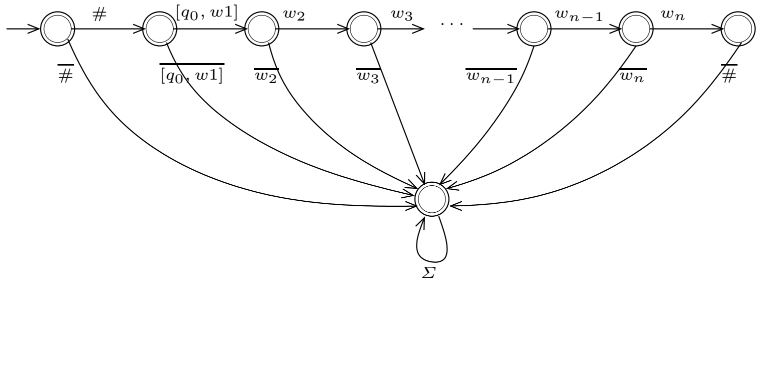
We start with an in-place Turing machine $M = (Q_M, \Sigma_M, \Gamma_M, \delta, q_0, \{q_f\})$ as described in Lemma 7.

We require $L(M)$ to be PSPACE-complete. For a given word w we have to determine whether $w \overset{?}{\in} L(M)$. Our transformation constructs an nfa A_w which accepts exactly those words x which are *not* legal sequences of configurations of M on input w ending with the unique accepting state q_f.

A_w uses the alphabet $\Sigma = (Q_M \times \Gamma_M) \cup \Gamma_M \cup \{\#\}$ to describe sequences of configurations of M separated by the new symbol $\#$. Every legal configuration has length exactly $|w|$ and is a word in $\Gamma_M^* \cdot (Q_M \times \Gamma_M) \cdot \Gamma_M^*$. The symbols $[q, a] \in Q_M \times \Gamma_M$ represent the head position of M on a cell with contents a while M is in state q.

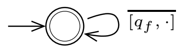
We can easily construct A_w, if we allow multiple initial states. Thus $L(A_w)$ is the union of the languages accepted by the automata described below. (It is an easy exercise to transform an autmaton with multiple initial states into an automaton with a unique initial state by at most doubling the number of transitions.)

(i) Accept every word x which does not start with $\#[q_0, w_1]w_2 \dots w_n\#$.



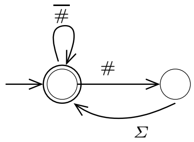
Here Σ means any symbol in Σ and $\overline{a}$ means any symbol in $\Sigma \setminus \{a\}$. Observe that the automaton rejects only if $\#$ is read in the rightmost state.

(ii) Accept every word x which does not contain $[q_f, \gamma]$ for any $\gamma \in \Gamma_M$.

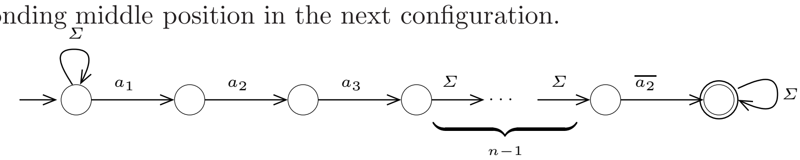


Read $\overline{[q_f, \cdot]}$ as any symbol in $\Sigma \setminus (\{q_f\} \times \Gamma_M)$.

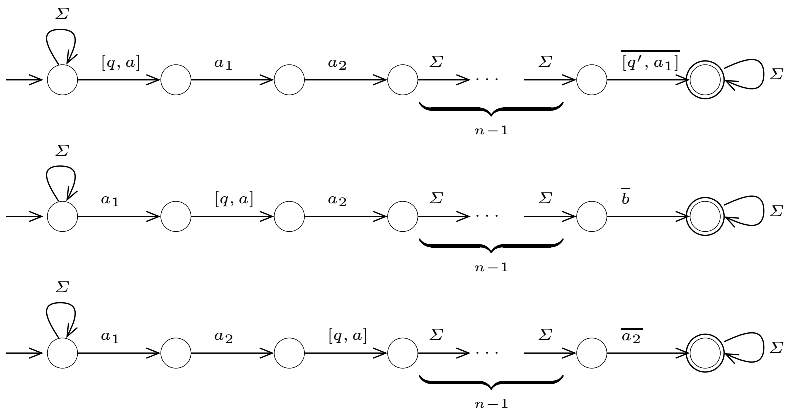
(iii) Accept every word x which does not end with $\#$.



(iv) In a legal sequence y of configurations, for every triple $y_{i-1}y_iy_{i+1} \in \Sigma^3$ of consecutive letters, the new middle symbol y_{i+n+1} is a function of $y_{i-1}y_iy_{i+1}$. Thus for any illegal sequence x of configurations there is a position i with $y_{i+n+1} \neq y_i$, if the head is not scanning y_i, or y_{i+n+1} is not updated correctly.
In particular, for each $a_1, a_2, a_3 \in \Gamma_M \cup \{\#\}$ accept every word x which does not have a_2 at the corresponding middle position in the next configuration.



Finally, for each $a_1, a_2 \in \Gamma_M$ and each $[q, a] \in Q_M \times \Gamma_M$ with $\delta(q, a) = (q', b, \rightarrow)$ for some $q' \in Q_M$ and some $b \in \Gamma_M$ accept wrong sequences:



Treat those $[q, a] \in Q_M \times \Gamma_M$ with $\delta(q, a) = (q', b, \leftarrow)$ accordingly.

The nfa A_w has $m \leq |w| \cdot 2 \cdot |\Sigma|^3$ states which is a constant multiple of $n = |w|$.

Observe that w belongs to $L(M)$ iff there is exactly one accepting computation x of M on w which is not accepted by A_w. (Remember, that M is deterministic.) Hence, if $w \notin L$, then $L(A_w) = \Sigma^*$ and the minimal equivalent nfa or regular expression has size 1. Moreover, if $w \in L$, then $L(A_w) = \Sigma^* \setminus \{x\}$ for some x which represents the accepting computation.

It suffices to show that $\Sigma^* \setminus \{x\}$ requires nfa's with at least n states. Any accepting computation has length at least 2^n, since M is a Turing-Machine as described in Lemma 7. Every dfa which excludes a single word of length 2^n needs at least 2^n states, thus every equivalent nfa needs at least n states. Hence, if M accepts w and thus $L(A_w) = \Sigma^* \setminus \{x\}$ for some x with $|x| \geq 2^n$, then every nfa which accepts $L(A_w)$ needs at least $n = |w|$ states.

Since determining whether w belongs to M is a PSPACE-complete problem, an approximation algorithm with an approximation factor of $o(m)$ solves a PSPACE-complete problem.

The inapproximability result for the number of transitions of nfa's and the number of symbols in regular expressions follows along the same lines. $\square$