Discrete Mathematics and Theoretical Computer Science DMTCS vol. **17:2**, 2015, 157–180

Classical Automata on Promise Problems[*]

Viliam Geffert[1,†] Abuzer Yakaryılmaz[2,‡]

[1] *Department of Computer Science, P. J. Šafárik University, Košice, Slovakia*
[2] *National Laboratory for Scientific Computing, Petrópolis, RJ, Brazil*

received 2014-10-14, *accepted 2015-09-05*.

Promise problems were mainly studied in quantum automata theory. Here we focus on state complexity of classical automata for promise problems. First, it was known that there is a family of unary promise problems solvable by quantum automata by using a single qubit, but the number of states required by corresponding one-way deterministic automata cannot be bounded by a constant. For this family, we show that even two-way nondeterminism does not help to save a single state. By comparing this with the corresponding state complexity of alternating machines, we then get a tight exponential gap between two-way nondeterministic and one-way alternating automata solving unary promise problems. Second, despite of the existing quadratic gap between Las Vegas realtime probabilistic automata and one-way deterministic automata for language recognition, we show that, by turning to promise problems, the tight gap becomes exponential. Last, we show that the situation is different for one-way probabilistic automata with two-sided bounded-error. We present a family of unary promise problems that is very easy for these machines; solvable with only two states, but the number of states in two-way alternating or any simpler automata is not limited by a constant. Moreover, we show that one-way bounded-error probabilistic automata can solve promise problems not solvable at all by any other classical model.

Keywords: descriptional complexity, promise problems, nondeterministic automata, probabilistic automata, alternating automata

1 Introduction

Promise problem is a generalization of language recognition. Instead of testing all strings over a given alphabet for language membership, we focus only on a given subset of strings as potential inputs that should be tested. The language under consideration and its complement form this subset.

Promise problems have already provided several different perspectives in computational complexity. (See the survey by Goldreich [18].) For example, it is not known whether the class of languages recognizable by bounded-error probabilistic polynomial-time algorithms has a complete problem, but the

[*] A preliminary version of this work was presented at the 16th International Workshop on Descriptional Complexity of Formal Systems (DCFS 2014), August 5–8, 2014, Turku, Finland [volume 8614 of *Lecture Notes in Computer Science*, pages 126–137, Springer-Verlag, 2014]. The ArXiv version is at http://arxiv.org/abs/1405.6671.

[†] Supported by the Slovak grant contracts VEGA 1/0142/15 and APVV-0035-10.

[‡] Supported by CAPES with grant 88881.030338/2013-01, ERC Advanced Grant MQC, and FP7 FET project QALGO. Moreover, the part of the research work was done while Yakaryılmaz was visiting Kazan Federal University.

class of promise problems solvable by the same type of algorithms has some complete problems. A similar phenomenon has been observed for quantum complexity classes [41]. The first known result on promise problems for restricted computational models we are aware of was given by Condon and Lipton in 1989 [8]. They showed that a promise version of emptiness problem is undecidable for one-way probabilistic finite automata. In the literature, some separation results regarding restricted computational models have also been given in the form of promise problems. (See, e.g., [11, 10].) The first result for restricted quantum models was given by Murakami et al. [32]: There exists a promise problem that can be solved by quantum pushdown automata exactly but not by any deterministic pushdown automaton. Recently, Ambainis and Yakaryılmaz [2] showed that there is an infinite family of promise problems which can be solved exactly just by tuning transition amplitudes of a one-way two-state quantum finite automaton (QFA), whereas the size of the corresponding classical automata grows above any fixed limit. Then, a series of papers [37, 13, 33, 34] presented many superiority results of QFAs over their classical counterparts. Meanwhile, some further results on succinctness of QFAs have been obtained in [44, 20, 21, 43, 4]. As a very recent study, different from the standard approach, Gruska et al. [19] focused on a subset of promise problems by introducing a new framework in which they extended the definition of the standard automaton so that the set of all accepting and rejecting states does not necessarily form the set of all states. Such automaton corresponds to a unique promise problem such that each yes-instance (no-instance) leads the automaton to an accepting (a rejecting) state.[(i)]

In this paper, we turn our attention to classical models and obtain some new results. In the next two sections, we give some preliminaries and present some basic facts for the classical models. Among others, we show that *(i)* the computational power of deterministic, nondeterministic, alternating, and Las Vegas probabilistic automata is the same, even after turning from the classical language recognition to solving promise problems, and, *(ii)* on promise problems, neither existing gaps of succinctness for classical language recognition can be violated, between any two of deterministic, nondeterministic, and alternating automata models.

Then, in Section 4, we consider a family of unary promise problems given by Ambainis and Yakaryılmaz in [2], solvable by only two-state one-way quantum finite automata. On the other hand, for solving this family by the use classical automata, we show that the exact number of states for one-way/two-way deterministic/nondeterministic automata is the same. Thus, for this family, replacing the classical one-way deterministic automata even by two-way nondeterministic automata does not help to save a single state. However, for the same family, already the one-way alternation does help, which gives us the tight exponential gap between one-way alternating and two-way sequential models of automata solving unary promise problems.

In Section 5, we show that the trade-off for the case of Las Vegas probabilistic versus one-way deterministic automata is different; by turning from language recognition to promise problems the tight quadratic gap changes to a tight exponential gap.

In Section 6, we show how the situation changes for finite state automata with two-sided bounded-error. First, we present a probabilistically very easy family of unary promise problems, i.e., solvable by one-way two-state automata with bounded-error. Then, we show that this family is hard for two-way alternating automata, and hence for any simpler classical automata as well. Finally, we prove that, on promise problems, bounded-error probabilistic automata are more powerful than any other classical model.

[(i)] In this way, for example, the class of promise problems defined by deterministic finite automata simply forms the set of all pairs of disjoint regular languages, i.e., $\{(\mathtt{P}_1, \mathtt{P}_2) \mid \mathtt{P}_1 \text{ and } \mathtt{P}_2 \text{ are disjoint regular languages}\}$.

2 Preliminaries

We assume the reader is familiar with the basic standard models of finite state automata (see e.g. [22]). For a more detailed exposition related to probabilistic automata, the reader is referred to [35, 5]. Here we only recall some models and introduce some elementary notation.

By Σ^*, we denote the set of strings over an alphabet Σ. The set of strings of length n is denoted by Σ^n and the unique string of length zero by ε. By $\mathcal{N}_+$, we denote the set of all positive integers. The cardinality of a finite set S is denoted by $\|S\|$.

A *one-way nondeterministic finite automaton* with ε-moves (1_εNFA, for short) is a quintuple $\mathcal{A} = (S, \Sigma, H, s_{\rm I}, S_{\rm A})$, where S is a finite set of states, Σ an input alphabet, $s_{\rm I} \in S$ an initial state, and $S_{\rm A} \subseteq S$ a set of accepting (final) states. Finally, $H \subseteq S \times (\Sigma \cup \{\varepsilon\}) \times S$ is a set of *transitions*, with the following meaning. $s \xrightarrow{a} s' \in H$: if the next input symbol is a, $\mathcal{A}$ gets from the state s to the state s' by reading a from the input. $s \xrightarrow{\varepsilon} s' \in H$: $\mathcal{A}$ gets from s to s' without reading any symbol from the input.[(ii)]

The machine $\mathcal{A}$ *halts* in the state s, if there are no executable transitions in H at a time. Typically, this happens after reading the entire input, but the computation may also be blocked prematurely, by *undefined transitions*, i.e., by absence of transitions for the current state s and the next input symbol a (or ε).

An *accepting computation* starts in the initial state $s_{\rm I}$ and, after reading the entire input, it halts in any accepting state $s' \in S_{\rm A}$. The language *recognized* (accepted) by $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of all input strings having at least one accepting computation. The inputs with no accepting computation are *rejected*.

A *two-way nondeterministic finite automaton* (2NFA) is defined in the same way as 1_εNFA, but now $\mathcal{A}$ can move in both directions along the input. For these reasons, the transitions in H are upgraded as follows. $s \xrightarrow{a\hookrightarrow} s'$, $s \xrightarrow{a\leftarrow} s'$, $s \xrightarrow{a\circlearrowright} s'$: if the current input symbol is a, $\mathcal{A}$ gets from s to s' and moves its input head one position to the right, to the left, or keeps it stationary, respectively. The input is enclosed in between two special symbols "$\vdash$" and "$\dashv$", called the *left and right endmarkers*, respectively. By definition, the input head cannot leave this area, i.e., there are no transitions moving to the left of $\vdash$ nor to the right of $\dashv$. $\mathcal{A}$ starts in $s_{\rm I}$ with the input head at the left endmarker and accepts by halting in $s' \in S_{\rm A}$ anywhere along the input tape.

A *deterministic finite automaton* (1_εDFA or 2DFA) can be obtained from 1_εNFA or 2NFA by claiming that the transition set H does not allow executing more than one possible transition at a time. Consequently, a 1_εDFA $\mathcal{A}$ can have at most one transition $s \xrightarrow{a} s'$, for each $s \in S$ and each $a \in \Sigma$. In theory, one can consider ε-transitions (even though this feature is rarely utilized): a transition $s \xrightarrow{\varepsilon} s'$ implies that there are no other transitions starting in the same state s. Similar restrictions can be formulated for 2DFAs.

A *self-verifying finite automaton* (1_εSVFA or 2SVFA) [25] is a 1_εNFA or 2NFA $\mathcal{A}$ which is additionally equipped with $S_{\rm R} \subseteq S$, the set of rejecting states disjoint from $S_{\rm A}$, the set of accepting states. The remaining states form $S_{\rm N} = S \setminus (S_{\rm A} \cup S_{\rm R})$, the set of neutral "don't know" states. In this model, *(i)* for each accepted input, $\mathcal{A}$ has at least one computation path halting in an accepting state and, moreover, no path halts in a rejecting state, *(ii)* for each rejected input, $\mathcal{A}$ has at least one computation path halting in a rejecting state and, moreover, no path halts in an accepting state. In both these cases, some computation paths may return "don't know" answers, by halting in neutral states or by getting into infinite loops.

(ii) Traditionally [22], the behavior is defined by a function $\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$. However, a traditional transition $\delta(s, a) \ni s'$ can be easily converted to $s \xrightarrow{a} s'$ and vice versa. In the same way, all models introduced here are consistent with the traditional definitions. Using transitions instead of a δ-function, we just make the notation more readable.

A *one-way probabilistic finite automaton* with ε-moves (1_εPFA) is defined in the same way as 1_εNFA, but the transitions in H are upgraded with probabilities, as follows. $s \xrightarrow{a,p} s' \in H$: if the current input symbol is a, $\mathcal{A}$ gets from the state s to the state s' by reading this symbol with probability $p \in [0,1]$. For $a = \varepsilon$, no input symbol is consumed. A transition with $p = 1$ may be displayed as $s \xrightarrow{a} s'$, since the next step is deterministic; transitions with $p = 0$ can be completely erased, as not executable. By definition,[(iii)] for each $s \in S$ and each $a \in \Sigma$, the sum of probabilities over all transitions beginning in s and labeled by $a' \in \{a, \varepsilon\}$ must be equal to 1. The overall accepting probability of $\mathcal{A}$ on a given input $w \in \Sigma^*$, denoted by $f_{\mathcal{A}}(w)$, is calculated over all accepting paths. Hence, the overall rejecting probability is $1 - f_{\mathcal{A}}(w)$.

A two-way version, 2PFA, was first introduced in [28].

A *Las Vegas probabilistic finite automaton* (Las Vegas 1_εPFA or Las Vegas 2PFA) [24] is a 1_εPFA or 2PFA of special kind, obtained from self-verifying automata (i.e., from nondeterministic automata of special kind). Thus, transitions are upgraded with probabilities, and the state set is partitioned into the sets of accepting, rejecting, and neutral "don't know" states. This gives three overall probabilities on a given input; accepting, rejecting, and neutral.

A *one-way alternating finite automaton* with ε-moves (1_εAFA) is obtained from 1_εNFA by partitioning the state set S into two disjoint subsets $S_\exists$ and $S_\forall$, the sets of *existential* and *universal* states, respectively. The global rules are defined as is usual for alternating devices (see e.g. [6]): if, at the given moment, there are several executable transitions, the machine $\mathcal{A}$ *(i)* nondeterministically chooses one of them, if it is in an existential state s, but *(ii)* follows, in parallel, all possible branches, if the current state s is universal.

For better readability, a nondeterministic switch from the existential s into one of the states $s_1, \ldots, s_k$, by reading an input symbol a, will be displayed as $s \xrightarrow{a} s_1 \vee \cdots \vee s_k$, while a parallel branching from the universal s to all these states as $s \xrightarrow{a} s_1 \wedge \cdots \wedge s_k$. The same applies for $a = \varepsilon$.

The input is accepted, if all branches in the nondeterministically chosen computation subtree, rooted in the initial state at the beginning of the input and embedded in the full tree of all possible computation paths, halt in accepting states at the end of the input.[(iv)]

Also this model was extended to a two-way version, 2AFA (see e.g. [15]).

A *realtime automaton* (rtDFA, rtSVFA, rtNFA, rtPFA, rtAFA,...) is the corresponding one-way finite automaton[(v)] that executes at most a constant number of ε-transitions in a row. (After that, the next input symbol is consumed or the machine halts.)

A *one-way ε-free automaton* (1DFA, 1SVFA, 1NFA, 1PFA, 1AFA,...) is the corresponding one-way finite automaton with no ε-transitions at all. (This is a special case of a realtime machine, with the number of executed ε-transitions bounded by zero.)

It should be pointed out that, in 1_εNFAs, the ε-transitions can be removed *without increasing* the number of states [29, Sect. 2.11]. Therefore, 1NFAs, rtNFAs, and 1_εNFAs agree in upper/lower bounds for states. The same works for 1SVFAs, rtSVFAs, and 1_εSVFAs, as well as for 1DFAs, rtDFAs, and 1_εDFAs. For this reason, we consider only 1NFAs, 1SVFAs, and 1DFAs.

(iii) Traditionally [36], the behavior of PFAs is defined by stochastic matrices.

(iv) The alternating automaton should not be confused with a *Boolean finite automaton*, which is (quite inappropriately) also sometimes called "alternating" in some literature. Instead of alternating existential and universal branching, the Boolean automaton can control acceptance by the use of arbitrarily complex Boolean functions. As an example, $s \xrightarrow{a} s_1 \wedge (s_2 \vee s_3)$ specifies that, for the current input symbol a, the subtree rooted in the state s is successful if and only if the subtree rooted in s_1 and at least one of those rooted in s_2, s_3 are successful.

(v) All realtime *quantum* models use a classical input head and so they are not allowed to create a superposition of the head positions on the input tape.

Occasionally, we shall also mention some restricted versions of two-way machines, e.g., *sweeping automata*, changing the direction of the input head movement only at the endmarkers [39, 27], or a very restricted version of sweeping automaton called a *restarting one-way automaton*, running one-way in an infinite loop [42] — if the computation does not halt at the right endmarker, then it jumps back to the initial state at the beginning of the input.

A *promise problem* is a pair of languages $\mathtt{P} = (\mathtt{P}_{\mathtt{yes}}, \mathtt{P}_{\mathtt{no}})$, where $\mathtt{P}_{\mathtt{yes}}, \mathtt{P}_{\mathtt{no}} \subseteq \Sigma^*$, for some Σ, and $\mathtt{P}_{\mathtt{yes}} \cap \mathtt{P}_{\mathtt{no}} = \emptyset$ [41].

The promise problem P is *solved* by a finite automaton $\mathcal{P}$, if $\mathcal{P}$ accepts each $w \in \mathtt{P}_{\mathtt{yes}}$ and rejects each $w \in \mathtt{P}_{\mathtt{no}}$. (That is, we do not have to worry about the outcome on inputs belonging neither to $\mathtt{P}_{\mathtt{yes}}$ nor to $\mathtt{P}_{\mathtt{no}}$.) If $\mathtt{P}_{\mathtt{yes}} \cup \mathtt{P}_{\mathtt{no}} = \Sigma^*$ and P is solved by $\mathcal{P}$, we have a classical language recognition, and say that $\mathtt{P}_{\mathtt{yes}}$ is *recognized* by $\mathcal{P}$.

If $\mathcal{P}$ is a probabilistic finite automaton, we say that $\mathcal{P}$ solves P with *error bound* $\varepsilon \in [0, \frac{1}{2})$, if $\mathcal{P}$ accepts each $w \in \mathtt{P}_{\mathtt{yes}}$ with probability at least $1-\varepsilon$ and rejects each $w \in \mathtt{P}_{\mathtt{no}}$ with probability at least $1-\varepsilon$. (The remaining probability goes to erroneous answers.) $\mathcal{P}$ solves P with a *bounded-error*, if, for some ε, it solves P with the error bound ε. If $\varepsilon = 0$, the problem is solved *exactly*.

A special case of bounded-error is a *one-sided bounded-error*, where either each $w \in \mathtt{P}_{\mathtt{yes}}$ is accepted with probability 1 or each $w \in \mathtt{P}_{\mathtt{no}}$ is rejected with probability 1.

If $\mathcal{P}$ is a Las Vegas probabilistic finite automaton, we say that $\mathcal{P}$ solves P with a *success probability* $p \in (0, 1]$, if *(i)* for each $w \in \mathtt{P}_{\mathtt{yes}}$, $\mathcal{P}$ accepts w with probability at least p and never rejects (i.e., the probability of rejection is 0) and *(ii)* for each $w \in \mathtt{P}_{\mathtt{no}}$, $\mathcal{P}$ rejects w with probability at least p and never accepts. (In both these cases, the remaining probability goes to "don't know" answers.)

3 Basic Facts on Classical Automata

We continue with some basic facts regarding classical automata. We start with a simple observation: The cardinality of the class of the promise problems defined by 1DFAs is uncountable. This holds even for the class defined by a fixed two-state 1DFA on unary languages. Let $\mathtt{U} \subseteq 1^*$ be any language defined over the unary input alphabet. Then, the following unary promise problem $\mathtt{P}(\mathtt{U})$ is solvable by a two-state 1DFA:

$$\begin{aligned} \mathtt{P}_{\mathtt{yes}}(\mathtt{U}) &= \{a^{2n} \mid n \in \mathtt{U}\}, \\ \mathtt{P}_{\mathtt{no}}(\mathtt{U}) &= \{a^{2n+1} \mid n \in \mathtt{U}\}. \end{aligned}$$

Clearly, $\mathtt{P}(\mathtt{U})$ can be solved by a 1DFA $\mathcal{D}$ with only two states, counting the length of the input modulo 2, switching between an accepting state and a rejecting state. Since there is a one-to-one correspondence between $\{\mathtt{U} \mid \mathtt{U} \subseteq 1^*\}$ and $\{\mathtt{P}(\mathtt{U}) \mid \mathtt{U} \subseteq 1^*\}$, the cardinality of the class of the promise problems solvable by $\mathcal{D}$ is uncountable. Moreover, if U is an uncomputable language, then both $\mathtt{P}_{\mathtt{yes}}(\mathtt{U})$ and $\mathtt{P}_{\mathtt{no}}(\mathtt{U})$ are uncomputable. The interested reader can easily find some other promise problems, solvable by very small 1DFAs, with yes- and no-instances satisfying some special properties (e.g., not context-free but context-sensitive, NP-complete, semi-decidable, and so on). One trivial construction for a binary language $\mathtt{L} \subseteq \{0,1\}^*$ is to define the promise problem as $\mathtt{P}(\mathtt{L}) = (\{0w \mid w \in \mathtt{L}\}, \{1w \mid w \in \mathtt{L}\})$. (A promise problem solvable by 1DFAs whose yes- and no-instances are not regular was previously given in [21].) Despite of the above facts, all trade-offs established between various models of automata stay valid also for promise problems.

First, let us show that the classes of promise problems solvable by deterministic, nondeterministic, alternating, and Las Vegas probabilistic finite automata are identical.

Theorem 3.1 *If a promise problem* $\mathtt{P} = (\mathtt{P}_{\mathtt{yes}}, \mathtt{P}_{\mathtt{no}})$ *can be solved by a* 2AFA $\mathcal{A}$ *with* n *states, it can also be solved by a* 1DFA $\mathcal{D}$ *with* $t(n) \leq 2^{n \cdot 2^n}$ *states.*

Proof: Let $\mathtt{R}$ denote the regular language recognized by $\mathcal{A}$. (That is, even though we are given a 2AFA $\mathcal{A}$ for solving a promise problem $\mathtt{P}$, such machine is still associated with some classical regular language $L(\mathcal{A}) = \mathtt{R}$.) The complement of this language will be denoted by $\overline{\mathtt{R}}$. Since the problem $\mathtt{P}$ is solvable by $\mathcal{A}$, we have that $\mathtt{P}_{\mathtt{yes}} \subseteq \mathtt{R}$ and $\mathtt{P}_{\mathtt{no}} \subseteq \overline{\mathtt{R}}$.

Now, by [30], each 2AFA $\mathcal{A}$ with n states can be transformed into an equivalent 1DFA $\mathcal{D}$ with $t(n) \leq 2^{n \cdot 2^n}$ states. This gives that $\mathtt{P}_{\mathtt{yes}} \subseteq \mathtt{R} = L(\mathcal{D})$ and $\mathtt{P}_{\mathtt{no}} \subseteq \overline{\mathtt{R}} = \overline{L(\mathcal{D})}$, that is, $\mathcal{D}$ accepts each $w \in \mathtt{P}_{\mathtt{yes}}$ and rejects each $w \in \mathtt{P}_{\mathtt{no}}$. Therefore, $\mathcal{D}$ can be used for solving $\mathtt{P}$. □

The proof of the above theorem can be easily updated for other classical models of automata — using the corresponding trade-off $t(n)$ for the conversion known from the literature — by which we obtain the following corollary.

Corollary 3.2 *Any trade-off* $t(n)$ *in the number of states for language recognition, between any two of deterministic, nondeterministic, or alternating automata models, is also a valid trade-off for promise problems.*

Next, we show that neither Las Vegas PFAs gain any computational power.

Theorem 3.3 *If a promise problem* $\mathtt{P} = (\mathtt{P}_{\mathtt{yes}}, \mathtt{P}_{\mathtt{no}})$ *can be solved by a Las Vegas* 2PFA $\mathcal{P}$ *with* n *states and any success probability* $p > 0$*, it can also be solved by a* 1DFA $\mathcal{D}$ *with* $t(n) \leq 2^{n^2+n}$ *states.*

Proof: First, for each $w \in \mathtt{P}_{\mathtt{yes}}$, the machine $\mathcal{P}$ has at least one accepting computation path and no rejecting path. Conversely, for each $w \in \mathtt{P}_{\mathtt{no}}$, the machine has at least one rejecting path and no accepting path. In both cases, the paths that are not successful return a "don't know" answer.

Therefore, by removing the probabilities and converting neutral "don't know" states into rejecting states, we obtain an n-state 2NFA $\mathcal{N}$ recognizing a regular language $\mathtt{R}$ such that $\mathtt{P}_{\mathtt{yes}} \subseteq \mathtt{R}$ and $\mathtt{P}_{\mathtt{no}} \cap \mathtt{R} = \emptyset$. Therefore, $\mathtt{P}_{\mathtt{no}} \subseteq \overline{\mathtt{R}}$.

Now, by [26], the 2NFA $\mathcal{N}$ can be converted to an equivalent 1DFA $\mathcal{D}$ with the number of states bounded by $t(n) \leq \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \binom{n}{i}\binom{n}{j}(2^i-1)^j \leq 2^{n^2+n}$, which gives $\mathtt{P}_{\mathtt{yes}} \subseteq \mathtt{R} = L(\mathcal{D})$ and $\mathtt{P}_{\mathtt{no}} \subseteq \overline{\mathtt{R}} = \overline{L(\mathcal{D})}$. That is, $\mathcal{D}$ accepts each $w \in \mathtt{P}_{\mathtt{yes}}$ and rejects each $w \in \mathtt{P}_{\mathtt{no}}$, and hence it can be used for solving $\mathtt{P}$. □

A similar idea can be used for Las Vegas 1_εPFAs:

Corollary 3.4 *If a promise problem* $\mathtt{P} = (\mathtt{P}_{\mathtt{yes}}, \mathtt{P}_{\mathtt{no}})$ *can be solved by a Las Vegas* 1_εPFA $\mathcal{P}$ *with* n *states and any success probability* $p > 0$*, it can also be solved by a* 1DFA $\mathcal{D}$ *with* $t(n) \leq 1+3^{(n-1)/3} \leq 2^{0.529 \cdot n}$ *states.*

Proof: By removing the probabilities, we first obtain an n-state 1_εSVFA $\mathcal{N}$ recognizing a regular language $\mathtt{R}$ such that $\mathtt{P}_{\mathtt{yes}} \subseteq \mathtt{R}$ and $\mathtt{P}_{\mathtt{no}} \subseteq \overline{\mathtt{R}}$. Then, after elimination of ε-transitions by the method described in [29, Sect. 2.11] (still not increasing the number of states), we have a 1SVFA that can be converted to an equivalent 1DFA $\mathcal{D}$ with $t(n) \leq 1 + 3^{(n-1)/3}$ states [25]. The resulting 1DFA is then used for solving $\mathtt{P}$. □

In Section 5, we show that the corresponding lower bound is also exponential.

4 A Classically Expensive Unary Promise Problem

Recently, Ambainis and Yakaryılmaz [2] presented a family of promise problems $\texttt{EvenOdd}(k)$, for $k \in \mathcal{N}_+$, where

$$\begin{aligned} \texttt{EvenOdd}_{\texttt{yes}}(k) &= \{a^{m \cdot 2^k} \mid m \text{ is even}\}, \\ \texttt{EvenOdd}_{\texttt{no}}(k) &= \{a^{m \cdot 2^k} \mid m \text{ is odd}\}, \end{aligned}$$

such that, for each k, the promise problem $\texttt{EvenOdd}(k)$ can be solved exactly by only a two-state realtime quantum finite automaton. On the other hand, for 1DFAs, 2^{k+1} states are necessary and also sufficient. Later, it was shown that 2^{k+1} states are also necessary in any bounded-error 1PFA [37] and that 2DFAs must use at least $2^{k+1}-1$ states [4].(vi)

Here we show that even two-way nondeterminism does not help us to save a single state for this promise problem; and hence 2^{k+1} states is the *exact* value for one-way/two-way deterministic/nondeterministic automata. After that, we show that alternation does help to reduce the number of states to $\Theta(k)$, already by the use of one-way ε-free machines. More precisely, $\texttt{EvenOdd}(k)$ can be solved by a 1AFA with $11k$–14 states (for $k \geq 3$); the corresponding lower bound for 1AFAs is $k+1$. For two-way machines, 2AFAs, the exact lower bound is an open problem, but we know that it must grow in k, which is an easy consequence of Theorems 3.1 and 4.1. Similarly, we do not know whether bounded-error 2PFAs can work with fewer than 2^{k+1} states.

The method used for proving the lower bound in the following theorem is not new — inspired by [40, 7, 14] — but it needs some modifications on promise problems.

Theorem 4.1 *For each $k \in \mathcal{N}_+$, any* 2NFA *for solving the promise problem* $\texttt{EvenOdd}(k)$ *needs at least* 2^{k+1} *states.*

Proof: Suppose, for contradiction, that the promise problem can be solved by a 2NFA $\mathcal{N}$ with $\|S\| < 2^{k+1}$ states.

Consider now the input $a^{2^{k+1}}$. Clearly, it must be accepted by $\mathcal{N}$, and hence there must exist at least one accepting computation path, so we can fix one such path. (Besides, being nondeterministic, $\mathcal{N}$ can have also other paths, not all of them necessarily accepting, but we do not care for them.) Along this fixed path, take the sequence of states

$$q_0,\ p_1, q_1,\ p_2, q_2,\ p_3, q_3, \ldots, p_r, q_r,\ p_{r+1}\,,$$

where q_0 is the initial state, p_{r+1} is an accepting state, and all the other states $\{p_i \cup q_i \mid 1 \leq i \leq r\}$ are at the left/right endmarkers of the input 2^{k+1}, such that:

- If p_i is at the left (resp., right) endmarker, then q_i is at the opposite endmarker, and the path from p_i to q_i traverses the entire input from left to right (resp., right to left), not visiting any of endmarkers in the meantime.
- The path between q_i and p_{i+1} starts and ends at the same endmarker, possibly visiting this endmarker several times, but not visiting the opposite endmarker. Such a path is called a "U-turn". This covers also the case of $q_i = p_{i+1}$ with zero number of executed steps in between.

(vi) Recently [1], it was also shown that the width of any nondeterministic or bounded-error stable OBBD cannot be smaller than 2^{k+1}, for a functional version of $\texttt{EvenOdd}(k)$.

Now, the path connecting p_i with q_i must visit all input tape positions in the middle of the input $a^{2^{k+1}}$, and hence $\mathcal{N}$ must enter 2^{k+1} states, at least one state per each input tape position. However, since $\|S\| < 2^{k+1}$, some state r_i must be repeated. That is, between p_i and q_i, there must exist a loop, starting and ending in the same state r_i, traveling some ℓ_i positions to the right, not visiting any of the endmarkers. For each traversal, we fix one such loop (even though the path from p_i to q_i may contain many such loops). Note that the argument for traveling across the input from right to left is symmetrical. Since this loop is in the middle of the traversal, we have that $\ell_i < 2^{k+1}$. But then ℓ_i can be expressed in the form

$$\ell_i \;=\; \gamma_i \cdot 2^{\alpha_i},$$

where $\gamma_i > 0$ is odd (not excluding $\gamma_i = 1$) and $\alpha_i \in \{0, 1, \ldots, k\}$. Note that if $\alpha_i \geq k+1$, we would have a contradiction with $\ell_i < 2^{k+1}$.

Now, consider the value

$$\ell \;=\; \mathrm{lcm}\{\ell_1, \ell_2, \ldots, \ell_r\},$$

the least common multiple of all fixed loops, for all traversals of the input $a^{2^{k+1}}$. Clearly, ℓ can also be expressed in the form

$$\ell \;=\; \gamma \cdot 2^{\alpha},$$

where $\gamma > 0$ is odd and $\alpha \in \{0, 1, \ldots, k\}$ (actually, $\gamma = \mathrm{lcm}\{\gamma_1, \ldots, \gamma_r\}$ and $\alpha = \max\{\alpha_1, \ldots, \alpha_r\}$).

We shall now show that the machine $\mathcal{N}$ must also accept the input

$$a^{2^{k+1} + \ell \cdot 2^{k-\alpha}}.$$

First, if q_i and p_{i+1} are connected by a U-turn path on the input $a^{2^{k+1}}$, they will be connected also on the input $a^{2^{k+1}+\ell \cdot 2^{k-\alpha}}$ since such path does not visit the opposite endmarker and $2^{k+1} + \ell \cdot 2^{k-\alpha} \geq 2^{k+1}$. Second, if p_i and q_i are connected by a left-to-right traversal along $a^{2^{k+1}}$, they will stay connected also along the input $a^{2^{k+1}+\ell \cdot 2^{k-\alpha}}$. This only requires to repeat the loop of the length ℓ_i beginning and ending in the state r_i. Namely, we can make $\frac{\ell}{\ell_i} \cdot 2^{k-\alpha}$ more iterations. Note that $\ell = \mathrm{lcm}\{\ell_1, \ldots, \ell_r\}$ and hence ℓ is divisible by ℓ_i, for each $i = 1, \ldots, r$, and that $\alpha \leq k$, which gives that $\frac{\ell}{\ell_i} \cdot 2^{k-\alpha}$ is an integer. Note also that these $\frac{\ell}{\ell_i} \cdot 2^{k-\alpha}$ additional iterations of the loop of the length ℓ_i travel exactly $(\frac{\ell}{\ell_i} \cdot 2^{k-\alpha}) \times \ell_i = \ell \cdot 2^{k-\alpha}$ additional positions to the right. Thus, if $\mathcal{N}$ has an accepting path for $a^{2^{k+1}}$, it must also have an accepting path for $a^{2^{k+1}+\ell \cdot 2^{k-\alpha}}$ (just by a straightforward induction on r). Therefore, $\mathcal{N}$ accepts $a^{2^{k+1}+\ell \cdot 2^{k-\alpha}}$. (Actually, $\mathcal{N}$ can have many other paths for this longer input, but they cannot rule out the accepting decision of the path constructed above.)

However, the input $a^{2^{k+1}+\ell \cdot 2^{k-\alpha}}$ should be rejected, since $2^{k+1} + \ell \cdot 2^{k-\alpha} = 2^{k+1} + \gamma \cdot 2^{\alpha} \cdot 2^{k-\alpha} = (2+\gamma) \cdot 2^k$, where γ is odd. This is a contradiction. So, $\mathcal{N}$ must have at least 2^{k+1} states. □

The above argument cannot be extended to alternating automata: using "cooperation" of several computation paths running in parallel, the number of states can be reduced from 2^{k+1} to $O(k)$, even by the use of one-way ε-free machines. We first present a conceptually simpler realtime version.

Lemma 4.2 *For each $k \in \mathcal{N}_+$, the language $\texttt{EvenOdd}_{\texttt{yes}}(k)$ can be recognized — and hence the promise problem $\texttt{EvenOdd}(k)$ can be solved — by an* rtAFA $\mathcal{A}$ *with $7k+2$ states.*

Proof: Recall that $\texttt{EvenOdd}_{\texttt{yes}}(k) = \{a^{m\cdot 2^k} \mid m \text{ is even}\}$, and hence the unary input string a^n is in this language if and only if n is an integer multiple of 2^{k+1}. In turn, this holds if and only if $b_{n,k} = b_{n,k-1} = \cdots = b_{n,0} = 0$, where $b_{n,i}$ denotes the i^{th} bit in the binary representation of the number n. (The bit positions are enumerated from right to left, beginning with zero for the least significant bit.) That is, written in binary, n should be of the form $\gamma 0^{k+1}$, for some $\gamma \in \{0,1\}^*$.

Based on this, the rtAFA $\mathcal{A}$ counts the length of the input modulo 2^{k+1} and verifies, for the binary representation of this length, that the last $k+1$ bits are all equal to zero. The important detail is that $\mathcal{A}$ counts *down*, starting from zero. That is, along the input, the value of the counter changes as follows:

$$\ell \;:=\; 0,\, 2^{k+1}{-}1,\, 2^{k+1}{-}2,\, \ldots,\, 2,\, 1,\, 0,\, 2^{k+1}{-}1,\, \ldots$$

Thus, the counter ℓ is interpreted as the number of input symbols not processed yet (and taken modulo 2^{k+1}). The machine $\mathcal{A}$ accepts if and only if this value is zero when it halts at the end of the input.

Implemented deterministically, this counting requires 2^{k+1} states. However, an alternating automaton can use several processes running in parallel. (See [38, 3] for a similar idea.) Each parallel process will keep only *one bit* of the current value of ℓ. That is, for $i \in \{k, k-1, \ldots, 0\}$, we shall use the states "i_0" and "i_1" to indicate that i^{th} bit in the current value of ℓ is cleared to zero or set to one, respectively. This will hold for each "proper" sequence of existential guesses along each input. The correctness of this guessing will be verified when $\mathcal{A}$ halts at the end of the input. (Actually, the alternating tree of computation paths will be highly redundant; the same bit will be remembered by a huge number of activated identical copies of i_0 or i_1, running in parallel.) Clearly, for each symbol a along the input, we need to decrement the counter ℓ by one and hence to update properly the corresponding bits. This is implemented by the use of some additional auxiliary states.

We are now ready to present the set of states for the rtAFA $\mathcal{A}$, together with their brief description:

i_x, for $i = k, \ldots, 0$ and $x \in \{0,1\}$: the i^{th} bit of ℓ is x $(b_{\ell,i} = x)$,

$i_{x,\forall_1}$, for $i = k, \ldots, 1$ and $x \in \{0,1\}$: the i^{th} bit of ℓ is x and, moreover, all bits to the right are set to 1 $(b_{\ell,i} = x$ and $b_{\ell,j} = 1$ for each $j < i)$,

$i_{x,\exists_0}$, for $i = k, \ldots, 1$ and $x \in \{0,1\}$: the i^{th} bit of ℓ is x and, moreover, at least one bit to the right is set to 0 $(b_{\ell,i} = x$ and $b_{\ell,j} = 0$ for some $j < i)$,

$i_{\exists_0}$, for $i = k, \ldots, 2$: the i^{th} bit of ℓ may be quite arbitrary, but at least one bit to the right is set to 0 $(b_{\ell,j} = 0$ for some $j < i)$,

s_{I}: the initial state.

All states of type i_0, i_1, $i_{\exists_0}$ are existential, whereas those of type $i_{0,\forall_1}$, $i_{1,\forall_1}$, $i_{0,\exists_0}$, $i_{1,\exists_0}$, s_{I} are universal. The states of type i_0, s_{I} are accepting and none of the remaining states is accepting.

Clearly, the total number of states is $2(k+1) + 2k + 2k + (k-1) + 1 = 7k+2$.

Transitions in $\mathcal{A}$ are defined as follows:

$i_x \xrightarrow{a} i_{x,\exists_0} \vee i_{1-x,\forall_1}$, for $i = k, \ldots, 1$ and $x \in \{0,1\}$: The i^{th} bit of ℓ is set to x if and only if either *(i)* the i^{th} bit of $\ell' = \ell{-}1$ is set to the same value x, provided that at least one bit to the right in ℓ' is set to zero, that is, $b_{\ell',j} = 0$ for some $j < i$, or *(ii)* the i^{th} bit of ℓ' is flipped to the complementary value $1-x$, provided that all bits the right in ℓ' are set to one, that is, $b_{\ell',j} = 1$ for each $j < i$. Therefore, in the state i_x, branching existentially while reading the next input symbol a, $\mathcal{A}$ guesses between switching to $i_{x,\exists_0}$ or to $i_{1-x,\forall_1}$.

$0_x \xrightarrow{a} 0_{1-x}$, for $x \in \{0,1\}$, a special case of i_x for $i = 0$: Transitions for the rightmost bit are simplified, since there is no bit position with $j < i$ and hence we do not need to use $0_{0,\forall_1}$, $0_{1,\forall_1}$, $0_{0,\exists_0}$, $0_{1,\exists_0}$. Therefore, in 0_x, the machine reads the next input symbol a deterministically, by switching to 0_{1-x}.

$i_{x,\forall_1} \xrightarrow{\varepsilon} i_x \wedge (i{-}1)_1 \wedge \cdots \wedge 0_1$, for $i = k, \ldots, 1$ and $x \in \{0,1\}$: In the auxiliary state $i_{x,\forall_1}$, the machine $\mathcal{A}$ has to verify the bit setting at the corresponding positions in ℓ', namely, that $b_{\ell',i} = x$ and that $b_{\ell',j} = 1$ for each $j < i$. Thus, branching universally with ε-transitions for each bit position $j \le i$, it splits into parallel processes $i_x, (i{-}1)_1, \ldots, 0_1$.

$i_{x,\exists_0} \xrightarrow{\varepsilon} i_x \wedge i_{\exists_0}$, for $i = k, \ldots, 2$ and $x \in \{0,1\}$: The role of the auxiliary state $i_{x,\exists_0}$ is similar to that of $i_{x,\forall_1}$, verifying the bit setting in ℓ'. This time we verify that *(i)* $b_{\ell',i} = x$ and that *(ii)* $b_{\ell',j} = 0$ for at least one $j < i$. In order not to mix universal and existential decisions, we delay the condition *(ii)* until the next computation step, and hence $\mathcal{A}$ branches universally in $i_{x,\exists_0}$, splitting into two parallel processes i_x and $i_{\exists_0}$, by the use of ε-transitions.

$1_{x,\exists_0} \xrightarrow{\varepsilon} 1_x \wedge 0_0$, for $x \in \{0,1\}$, a special case of $i_{x,\exists_0}$ for $i = 1$: Transitions are simplified, since we can work without $1_{\exists_0}$. Therefore, in $1_{x,\exists_0}$, the machine splits universally into 1_x and 0_0, using ε-transitions.

$i_{\exists_0} \xrightarrow{\varepsilon} (i{-}1)_0 \vee \cdots \vee 0_0$, for $i = k, \ldots, 2$: In the auxiliary state $i_{\exists_0}$, we verify that $b_{\ell',j} = 0$ for at least one $j < i$. Therefore, branching existentially with ε-transitions, $\mathcal{A}$ guesses which bit is set to zero, switching to j_0, for some $j < i$.

$s_1 \xrightarrow{a} k_{1,\forall_1}$: At the very beginning, $\ell = n$. Therefore, for each accepted input a^n, the initial value of ℓ must be an integer multiple of 2^{k+1}, with the last $k{+}1$ bits all equal to zero. Thus, after reading the first input symbol a, we should get $\ell' = \ell{-}1$ in which all these bits are set to one. For this reason, in the initial state, the machine reads the first input symbol a by switching deterministically to $k_{1,\forall_1}$.

It is easy to see from the above transitions that $\mathcal{A}$ is a realtime machine: After reading an input symbol, a computation path can pass through at most two ε-transitions in a row, after which it gets to a state i_x, for some $i = k, \ldots, 0$ and some $x \in \{0,1\}$. In i_x, the machine waits for a next input symbol.

To show that $\mathcal{A}$ indeed recognizes $\texttt{EvenOdd}_{\texttt{yes}}(k)$, we need to prove the following claim.

Claim. *For each $\ell \ge 0$, each $i = k, \ldots, 0$, and each $x \in \{0,1\}$, the* rtAFA *$\mathcal{A}$ has an accepting alternating subtree of computation paths rooted in the state i_x at the input position ℓ (measured from the end of the input) if and only if the i^{th} bit of ℓ is equal to x.*

The claim is proved by induction on $\ell = 0, 1, 2, \ldots$

First, let $\ell = 0$, that is, $\mathcal{A}$ has already processed all input symbols. Clearly, all bits of ℓ are equal to zero, i.e., $b_{\ell,i} = 0$ for each i. Now, since there are no more input symbols to be processed and there are no ε-transitions starting in the state i_x, the machine $\mathcal{A}$ must halt in i_x. Therefore, $\mathcal{A}$ has an accepting alternating subtree rooted in i_x at the end of the input if and only if i_x is an accepting state. By definition of accepting states in $\mathcal{A}$ (the states of type i_0 are accepting but those of type i_1 are not), this holds if and only if $x = 0$, which in turn holds if and only if $x = 0 = b_{\ell,i}$, the i^{th} bit of ℓ.

Second, by induction, assume that the claim holds for $\ell' = \ell - 1$. Now, by definition of the transitions in the state i_x for $i > 0$, taking into account subsequent ε-transitions passing through auxiliary states, it can be seen[(vii)] that $\mathcal{A}$ has an accepting alternating subtree of computation paths rooted in the state i_x at the input position ℓ if and only if either *(i)* $\mathcal{A}$ has an accepting alternating subtree rooted in the state i_x at the input position $\ell' = \ell - 1$ and, moreover, the same holds for at least one subtree rooted in a state j_0 at the position ℓ', for some $j < i$, or *(ii)* $\mathcal{A}$ has an accepting alternating subtree rooted in i_{1-x} at the input position ℓ' and, moreover, the same holds for all subtrees rooted in j_1 at the position ℓ', for each $j < i$. However, by induction, using $\ell' = \ell - 1$, this statement holds if and only if either *(i)* $b_{\ell',i} = x$ and, moreover, $b_{\ell',j} = 0$, for some $j < i$, or *(ii)* $b_{\ell',i} = 1 - x$ and, moreover, $b_{\ell',j} = 1$, for each $j < i$. Using $\ell = \ell' + 1$, it is easy to see that this holds if and only if $b_{\ell,i} = x$.

This completes the proof of the claim.

Now, recall that the machine $\mathcal{A}$ reads the first input symbol a by switching deterministically from the initial state to the state $k_{1,\forall_1}$. After that, branching universally with ε-transitions for each bit position $j \leq k$, the computation splits into parallel processes $k_1, (k-1)_1, \ldots, 0_1$. Thus, using the above claim, $\mathcal{A}$ accepts a^n with $n > 0$ if and only if, in $\ell = n - 1$, the last $k+1$ bits are all equal to 1. Clearly, this holds if and only if the last $k+1$ bits in n are all equal to 0, that is, if and only if n is and integer multiple of 2^{k+1}. This gives that the language recognized by $\mathcal{A}$ agrees with $\texttt{EvenOdd}_{\texttt{yes}}(k)$ on all inputs of length $n > 0$.

For the special case of $n = 0$, we have that $a^0 = a^{0 \cdot 2^k} \in \texttt{EvenOdd}_{\texttt{yes}}(k)$, since $m = 0$ is even. However, $a^0 = \varepsilon$ is accepted by $\mathcal{A}$ due to the fact that the initial state s_{I} is also an accepting state, which completes the argument. □

By increasing slightly the number of states, we can replace the realtime alternating machine from the above lemma by a one-way AFA not using any ε-transitions:

Theorem 4.3 *For each $k \geq 3$, the language $\texttt{EvenOdd}_{\texttt{yes}}(k)$ can be recognized — and hence the promise problem $\texttt{EvenOdd}(k)$ can be solved — by a* 1AFA *$\mathcal{A}''$ with $11k - 14$ states.*

Proof: By inspection of the rtAFA $\mathcal{A}$ constructed in Lemma 4.2, it can be seen that $\mathcal{A}$ can read an input symbol only by transitions starting in the states of type i_x, s_{I}. After reading, the computation path can pass through at most two subsequent ε-transitions until it gets to a state of type i_x, ready to read from the input again.

First, as an intermediate product, we replace $\mathcal{A}$ by an equivalent machine $\mathcal{A}'$ in which reading an input symbol is always followed by *exactly three* subsequent ε-transitions, along each computation path. This only requires to delay artificially the moment when the automaton gets to a state of type i_x by the use of some additional states, namely, the following additional states:

(vii) With obvious modifications, the argument presented here can be easily extended also for the case of $i = 0$ (the rightmost bit), which we leave to the reader.

i'_x, i''_x, for $i = k, \ldots, 0$ and $x \in \{0,1\}$: delay switching to the state i_x by one or two steps, respectively — implemented by transitions $i''_x \xrightarrow{\varepsilon} i'_x \xrightarrow{\varepsilon} i_x$,

$0'''_x$, for $x \in \{0,1\}$: delay switching to the state 0_x by three steps — implemented by $0'''_x \xrightarrow{\varepsilon} 0''_x$.

Taking also into account the states used by the original machine $\mathcal{A}$, the total number of states increases to $(7k+2) + 4(k+1) + 2 = 11k+8$.

Transitions for the "original" states (cf. construction in Lemma 4.2) are modified as follows:

$i_x \xrightarrow{a} i_{x,\exists_0} \vee i_{1-x,\forall_1}$, for $i = k, \ldots, 1$ and $x \in \{0,1\}$ (not modified),

$0_x \xrightarrow{a} 0'''_{1-x}$, for $x \in \{0,1\}$,

$i_{x,\forall_1} \xrightarrow{\varepsilon} i''_x \wedge (i-1)''_1 \wedge \cdots \wedge 0''_1$, for $i = k, \ldots, 1$ and $x \in \{0,1\}$,

$i_{x,\exists_0} \xrightarrow{\varepsilon} i''_x \wedge i_{\exists_0}$, for $i = k, \ldots, 2$ and $x \in \{0,1\}$,

$1_{x,\exists_0} \xrightarrow{\varepsilon} 1''_x \wedge 0''_0$, for $x \in \{0,1\}$,

$i_{\exists_0} \xrightarrow{\varepsilon} (i-1)'_0 \vee \cdots \vee 0'_0$, for $i = k, \ldots, 2$,

$s_{\mathrm{I}} \xrightarrow{a} k_{1,\forall_1}$ (not modified).

Since $\mathcal{A}'$ and $\mathcal{A}$ share the same set of accepting states, it should be obvious that they recognize the same language.

Second, replace all ε-transitions in $\mathcal{A}'$ by transitions reading the input symbol a. That is, each transition of type $s_1 \xrightarrow{\varepsilon} s_2$ is replaced by $s_1 \xrightarrow{a} s_2$; the transitions already reading input symbols do not change; nor does the set of accepting states. This way we obtain a one-way AFA $\mathcal{A}''$ without any ε-transitions such that, for each $n \geq 0$, it accepts the input a^{4n} if and only if $\mathcal{A}'$ accepts a^n, i.e., if and only if $a^n \in \mathtt{EvenOdd}_{\mathtt{yes}}(k)$. The new machine $\mathcal{A}''$ does not accept any input the length of which is not an integer multiple of 4.

Therefore, with $11k+8$ states, $\mathcal{A}''$ recognizes the language

$$\{(a^4)^{m \cdot 2^k} \mid m \text{ is even}\} = \{a^{m \cdot 2^{k+2}} \mid m \text{ is even}\} = \mathtt{EvenOdd}_{\mathtt{yes}}(k+2)\,.$$

Using the substitution $k' = k+2$, we thus get that $\mathtt{EvenOdd}_{\mathtt{yes}}(k')$ can be recognized by a one-way ε-free AFA $\mathcal{A}''$ with $11(k'-2) + 8 = 11k'-14$ states, for each $k' \geq 3$. □

To show that the gap exhibited by Theorems 4.1 and 4.3 is tight, we need an upper bound for converting alternation into determinism on unary promise problems:

Lemma 4.4 *If a promise problem* $\mathtt{P} = (\mathtt{P}_{\mathtt{yes}}, \mathtt{P}_{\mathtt{no}})$ *built over a unary input alphabet can be solved by a* 1AFA *with* n *states, it can also be solved by a* 1DFA *with* $t(n) \leq 2^n$ *states.*

Proof: As pointed out in [31, Sect. 5], each 1AFA $\mathcal{A}$ with n states recognizing a unary language $\mathtt{U}$ can be replaced by an equivalent 1DFA $\mathcal{D}$ with 2^n states. This is a simple consequence of the fact that $\mathcal{A}$ can be replaced by a 1DFA $\mathcal{D}$ with 2^n states recognizing $\mathtt{U}^{\mathrm{R}}$, the reversal of the original language [6]. (Actually, the construction in [6] works for Boolean automata — see Footnote (iv) — which covers, as a special case, the classical alternating automata as well.) However, for unary languages, we have $\mathtt{U}^{\mathrm{R}} = \mathtt{U}$.

Then, by the same reasoning as used in the proof of Theorem 3.1, this trade-off easily extends to unary promise problems. □

By the above lemma, the lower bound for solving $\texttt{EvenOdd}(k)$ by 1AFAs is $k+1$ states, since a 1AFA with less than $k+1$ states would give us a 1DFA with less than 2^{k+1} states for this promise problem, which contradicts Theorem 4.1.

By combining Theorems 4.1 and 4.3 with Lemma 4.4, we thus get:

Corollary 4.5 *The tight gap of succinctness between two-way* NFA*s and one-way ε-free* AFA*s is asymptotically exponential on promise problems, even for a unary input alphabet.*

5 Las Vegas Probabilistic Finite Automata

In the case of language recognition, the gap of succinctness between Las Vegas realtime PFAs and one-way DFAs can be at most quadratic and this gap is tight [9, 24]. Quite surprisingly, despite of this fact and despite of the fact that the existing gaps for language recognition are not be violated on promise problems by classical models of automata (cf. Corollary 3.2), we shall show here that this does not hold for the case of one-way Las Vegas probabilistic versus one-way deterministic automata; by switching from language recognition to promise problems, the quadratic gap increases to a tight exponential gap.

For this purpose, we introduce a family of promise problems solvable by Las Vegas 1PFAs with a linear number of states and success probability arbitrarily close to 1, but for which any 1DFAs require an exponential number of states.(viii)

Consider a family of promise problems $\texttt{TRIOS}(n, r)$, for $n, r \in \mathcal{N}_+$, such that

$$\begin{aligned}
\texttt{TRIOS}_{\texttt{yes}}(n,r) &= \{\, \#x_1x_1y_1\#x_2x_2y_2\cdots\#x_rx_ry_r \mid x_1,y_1,\ldots,x_r,y_r \in \{0,1\}^n, \\
&\qquad \forall i \in \{1,\ldots,r\}\ \exists j \in \{1,\ldots,n\} : x_{i,j} < y_{i,j} \,\}, \\
\texttt{TRIOS}_{\texttt{no}}(n,r) &= \{\, \#x_1y_1x_1\#x_2y_2x_2\cdots\#x_ry_rx_r \mid x_1,y_1,\ldots,x_r,y_r \in \{0,1\}^n, \\
&\qquad \forall i \in \{1,\ldots,r\}\ \exists j \in \{1,\ldots,n\} : x_{i,j} > y_{i,j} \,\},
\end{aligned}$$

where $x_{i,j}, y_{i,j} \in \{0,1\}$ denote the j^{th} bit in the respective strings $x_i, y_i \in \{0,1\}^n$.

Theorem 5.1 *For each $n, r \in \mathcal{N}_+$, the promise problem $\texttt{TRIOS}(n, r)$ can be solved by a Las Vegas* 1PFA *$\mathcal{P}$ with $4n+3$ states and success probability $1 - (\frac{n-1}{n})^r$.*

Proof: The automaton $\mathcal{P}$ uses the following state set:

$$\begin{aligned}
Q &= \{q_{\text{I}}, q_{\text{A}}, q_{\text{R}}\} \cup \{s_i \mid i \in \{1,\ldots,n\}\} \cup \\
&\quad \{t_{i,0} \mid i \in \{1,\ldots,2n\}\} \cup \{t_{i,1} \mid i \in \{1,\ldots,n\}\},
\end{aligned}$$

where q_{I} is the initial and also the "don't know" state, while $q_{\text{A}}, q_{\text{R}}$ are the accepting and rejecting states, respectively.

(viii) Remark that, in [17], we argued the same exponential separation by using some other families of promise problems, but it appeared that some of these families can also be solved by 1DFAs with a linear number of states, and hence they cannot be used as witness families for proving such separation. More precisely, a single sentence statement given just before Corollary 4 in [17] is not correct.

Recall that we do not have to worry about the outcome of the computation if the input is not of the form $(\#\{0,1\}^{3n})^r$. Now, let $\#x_iu_iv_i$, with $x_i, u_i, v_i \in \{0,1\}^n$, denote the i^{th} input segment. Typically, for $i > 1$, the processing of the i^{th} segment is activated already somewhere in the middle of the previous segment, by switching to the initial state q_{I}. For $i = 1$, the machine begins with the first symbol $\#$. For these reasons, in the state q_{I}, the machine ignores all symbols along the input until it reads the separator $\#$, when it enters the state s_1. This is implemented by transitions $q_{\text{I}} \xrightarrow{b} q_{\text{I}}$, for each $b \in \{0,1\}$, together with $q_{\text{I}} \xrightarrow{\#} s_1$. (Unless otherwise stated explicitly, all transitions throughout this proof are "deterministic", i.e., with probability 1.)

Next, starting in the state s_1, the automaton $\mathcal{P}$ moves along the string x_i and chooses, with equal probability $\frac{1}{n}$, one of the symbols $x_{i,1}, \ldots, x_{i,n} \in \{0,1\}$ for subsequent verification. This is implemented as follows. If $\mathcal{P}$ is in the state s_j, for $j \in \{1, \ldots, n\}$, and the next input symbol is $b \in \{0,1\}$, then $\mathcal{P}$ reads this symbol by switching to the state $t_{1,b}$ with probability $p_j = \frac{1}{p'_1 \cdots p'_{j-1}} \cdot \frac{1}{n}$ and to the state s_{j+1} with probability $p'_j = 1 - p_j$. (As usual in mathematical notation, we take $p'_1 \cdots p'_{j-1} = 1$ for $j = 1$.) The reader may easily verify that the sequence of probabilities $p_1, \ldots, p_n$ ends with $p_n = 1$, which gives $p'_n = 0$, and hence there is no need to introduce a state s_{n+1}. It is also easy to see that $\mathcal{P}$ gets from the state s_1 at the beginning of $x_i = x_{i,1} \cdots x_{i,n}$ to the state $t_{1,x_{i,j}}$ positioned just behind the symbol $x_{i,j}$ with probability $\frac{1}{n}$.

The next phase depends on whether $\mathcal{P}$ is in the state $t_{1,0}$ or in $t_{1,1}$, that is, on whether the chosen bit $x_{i,j}$ is equal to 0 or to 1.

For the case of $x_{i,j} = 0$, the routine starting in the state $t_{1,0}$ is responsible to verify that $v_{i,j} = 1$, which corresponds to the case of $\#x_iu_iv_i = \#x_ix_iy_i$, with $x_{i,j} < y_{i,j}$. Since $\mathcal{P}$ gets to $t_{1,0}$ just after reading the symbol $x_{i,j}$, this only requires to examine the bit that is placed exactly $2n$ positions to the right. This is implemented by transitions $t_{j,0} \xrightarrow{b} t_{j+1,0}$, for each $j \in \{1, \ldots, 2n-1\}$ and each $b \in \{0,1\}$. Finally, if $v_{i,j} = 1$, then $\mathcal{P}$ accepts, using $t_{2n,0} \xrightarrow{1} q_{\text{A}}$. Otherwise, $\mathcal{P}$ switches back to the initial state, by $t_{2n,0} \xrightarrow{0} q_{\text{I}}$. In the accepting state q_{A}, the machine just consumes the remaining part of the input, using $q_{\text{A}} \xrightarrow{a} q_{\text{A}}$, for each $a \in \{0,1,\#\}$. Conversely, in the initial state q_{I}, the machine proceeds to examine the next input segment $\#x_{i+1}u_{i+1}v_{i+1}$, by transitions already described above. However, if $\mathcal{P}$ switches to q_{I} while processing the last input segment (for $i = r$), it halts in q_{I} at the end of the input. This is interpreted as a "don't know" answer.

The case of $x_{i,j} = 1$ is very similar. This time the routine starting in the state $t_{1,1}$ verifies that $u_{i,j} = 0$, which corresponds to the case of $\#x_iu_iv_i = \#x_iy_ix_i$, with $x_{i,j} > y_{i,j}$. This only requires to examine the bit placed exactly n positions to the right of the symbol $x_{i,j}$. This is implemented by transitions $t_{j,1} \xrightarrow{b} t_{j+1,1}$, for each $j \in \{1, \ldots, n-1\}$ and each $b \in \{0,1\}$. After that, if $u_{i,j} = 0$, then $\mathcal{P}$ rejects, using $t_{n,1} \xrightarrow{0} q_{\text{R}}$. Otherwise, $\mathcal{P}$ switches back to the initial state, by $t_{n,1} \xrightarrow{1} q_{\text{I}}$. In the rejecting state q_{R}, the machine consumes the rest of the input, using $q_{\text{R}} \xrightarrow{a} q_{\text{R}}$, for each $a \in \{0,1,\#\}$. Conversely, in the initial state q_{I}, the machine proceeds in the same way as in the case of $x_{i,j} = 0$; either it proceeds to the next input segment, or else it halts in q_{I} at the end of the input, giving a "don't know" answer.

Summing up, if the i^{th} segment is of the form $\#x_iu_iv_i = \#x_ix_iy_i$, with $x_{i,j} < y_{i,j}$ for some $j \in \{1, \ldots, n\}$, then $\mathcal{P}$, starting to process this segment in the state q_{I}, accepts the entire input with probability at least $\frac{1}{n}$ (the exact value increases in the number of bit pairs satisfying $x_{i,j} < y_{i,j}$) and proceeds to the next segment in the state q_{I} with probability at most $1 - \frac{1}{n}$. (The machine never rejects on such segment, since $u_i = x_i$ and there are no bit pairs satisfying $x_{i,j} > u_{i,j}$.)

Now, recall that each $w \in \texttt{TRIOS}_{\texttt{yes}}(n,r)$ consists of r segments having this form, and hence $\mathcal{P}$ gives the "don't know" answer with probability at most $(1-\frac{1}{n})^r$ for w. Therefore, w is accepted with probability at least $1-(1-\frac{1}{n})^r$ and never rejected.

Conversely, if i^{th} segment is of the form $\#x_iu_iv_i = \#x_iy_ix_i$, with $x_{i,j} > y_{i,j}$ for some $j \in \{1,\ldots,n\}$, then, in the course of processing this segment, $\mathcal{P}$ rejects the entire input with probability at least $\frac{1}{n}$ and proceeds to the next segment with probability at most $1-\frac{1}{n}$. (Here the machine never accepts, since $v_i = x_i$ and there are no bit pairs with $x_{i,j} < v_{i,j}$.)

But then each $w \in \texttt{TRIOS}_{\texttt{no}}(n,r)$, consisting of r segments of this form, is rejected with probability at least $1-(1-\frac{1}{n})^r$ and never accepted, by the same reasoning as for $\texttt{TRIOS}_{\texttt{yes}}(n,r)$. □

It is obvious that, for arbitrarily small fixed constant $\varepsilon > 0$, by taking $r = \lceil \log\varepsilon / \log(\frac{n-1}{n}) \rceil$, we shall obtain a family of promise problems, namely, $\texttt{TRIOS}'_\varepsilon(n)$, for $n \in \mathcal{N}_+$, where $\texttt{TRIOS}'_\varepsilon(n) = \texttt{TRIOS}(n, \lceil \log\varepsilon / \log(\frac{n-1}{n}) \rceil)$, such that the n^{th} member of this family can be solved by a Las Vegas 1PFA with $4n+3$ states and success probability at least $1-\varepsilon$.

On the other hand, for each fixed $r \in \mathcal{N}_+$, 1DFAs need an exponential number of states for solving $\texttt{TRIOS}(n,r)$.

Theorem 5.2 *For each $n, r \in \mathcal{N}_+$, any* 1DFA *needs at least 2^n states for solving the promise problem* $\texttt{TRIOS}(n,r)$.

Proof: For contradiction, suppose that this promise problem can be solved by a 1DFA $\mathcal{D}$ using $\|S\| < 2^n$ states. Let s_{I} be its initial state, and let $\delta(q,u) \in S$ denote the state reached by $\mathcal{D}$ from the state $q \in S$ by reading the string $u \in \Sigma^*$. Clearly, $\delta(q,uv) = \delta(\delta(q,u),v)$, for each state q and each two strings u, v.

Now, for each state $q \in S$, consider the enumeration of states $\delta(q,\#u)$, running over all strings $u \in \{0,1\}^n$. Since the number of states in S is smaller than the number of strings in $\{0,1\}^n$, we have, for each $q \in S$, that there must exist at least two different strings $u, v \in \{0,1\}^n$ with $\delta(q,\#u) = \delta(q,\#v)$. Using some lexicographic ordering for $\{0,1\}^n$, we can pick u_q and v_q, the first pair of strings satisfying $u_q \neq v_q$ and $\delta(q,\#u_q) = \delta(q,\#v_q)$. Moreover, without loss of generality, we can assume that, for some bit position $j \in \{1,\ldots,n\}$, we have $u_{q,j} < v_{q,j}$. (Here $u_{q,j}, v_{q,j}$ denote the j^{th} bit in the respective strings u_q, v_q.) Otherwise, by renaming, the roles of u_q and v_q can be swapped.

Consider now the input

$$\begin{array}{lcl} w_{\text{A}} & = & \#u_{q_1}u_{q_1}v_{q_1}\#u_{q_2}u_{q_2}v_{q_2}\cdots\#u_{q_r}u_{q_r}v_{q_r}\,, \text{ where} \\ q_1 & = & s_{\text{I}}\,, \\ q_{i+1} & = & \delta(q_i, \#u_{q_i}u_{q_i}v_{q_i})\,, \text{ for } i = 1,\ldots,r. \end{array}$$

It should be easily seen that, along the input w_{A}, $\mathcal{D}$ passes the segment boundaries by the sequence of states $q_1, q_2, \ldots, q_r, q_{r+1}$. More precisely, for each $i = 1,\ldots,r$, the machine is in the state q_i at the beginning of the i^{th} segment $\#u_{q_i}u_{q_i}v_{q_i}$, halting in the state q_{r+1} at the end of the last segment. It is also obvious that $w_{\text{A}} \in \texttt{TRIOS}_{\texttt{yes}}(n,r)$, since, for each segment $\#u_{q_i}u_{q_i}v_{q_i}$, we have a bit position $j \in \{1,\ldots,n\}$ satisfying $u_{q,j} < v_{q,j}$. Therefore, q_{r+1} is an accepting state.

Next, consider the input

$$w_{\text{R}} \;=\; \#v_{q_1}u_{q_1}v_{q_1}\#v_{q_2}u_{q_2}v_{q_2}\cdots\#v_{q_r}u_{q_r}v_{q_r}\,.$$

Clearly, also here the computation starts in the state $q_1 = s_{\rm I}$. Now, using the fact that $\delta(q, \#u_q) = \delta(q, \#v_q)$ for each state $q \in S$, we have that q_{i+1} can also be expressed as

$$\begin{aligned} q_{i+1} &= \delta(q_i, \#u_{q_i}u_{q_i}v_{q_i}) = \delta(\delta(q_i, \#u_{q_i}), u_{q_i}v_{q_i}) \\ &= \delta(\delta(q_i, \#v_{q_i}), u_{q_i}v_{q_i}) = \delta(q_i, \#v_{q_i}u_{q_i}v_{q_i})\,, \end{aligned}$$

for each $i = 1, \ldots, r$. Therefore, along the input $w_{\rm R}$, $\mathcal{D}$ passes the segment boundaries by the same sequence of states $q_1, q_2, \ldots, q_r, q_{r+1}$, halting in q_{r+1}. But then, since q_{r+1} is an accepting state, $\mathcal{D}$ accepts $w_{\rm R}$. However, $w_{\rm R} \in \texttt{TRIOS}_{\texttt{no}}(n, r)$, since, for each segment $\#v_{q_i}u_{q_i}v_{q_i}$, we have a bit position $j \in \{1, \ldots, n\}$ satisfying $v_{q,j} > u_{q,j}$.

Thus, each 1DFA $\mathcal{D}$ with less than 2^n states accepts an input that should be rejected, a contradiction. □

As a consequence of the above theorem, we get the asymptotically optimal exponential size $\Theta(2^n)$ for 1DFAs solving $\texttt{TRIOS}(n, r)$; the construction of a 1DFA with $O(2^n)$ states is quite straightforward.

On the other hand, Theorem 5.1 guarantees $4n{+}3$ states for Las Vegas 1PFAs solving $\texttt{TRIOS}(n, r)$, and hence we have an exponential lower bound for the gap between these two models of automata. Finally, by Corollary 3.4, we have an exponential upper bound, which gives:

Corollary 5.3 *The tight gap of succinctness between one-way* DFA*s and Las Vegas one-way ε-free* PFA*s is asymptotically exponential on promise problems.*

Due to Theorem 5.1, we know that $\texttt{TRIOS}(n, n^2)$ can be solved by a Las Vegas 1PFA with $4n{+}3$ states and success probability $\sigma(n) \geq 1 - (1{-}\frac{1}{n})^{n^2}$. Thus, using the fact [23, Lm. A.3.60+] that $(1{-}\frac{1}{x})^x < \frac{1}{e}$ for each real $x > 1$, the probability of a "don't know" outcome is at most

$$1{-}\sigma(n) \leq \left(1{-}\tfrac{1}{n}\right)^{n\cdot n} < \left(\tfrac{1}{e}\right)^n = \tfrac{1}{e^n}\,.$$

This Las Vegas 1PFA can be converted to an equivalent restarting one-way PFA, executing the procedure in an infinite loop, by restarting the entire computation when the original machine halts in the "don't know" state at the end of the input. (This can be easily achieved by using a single additional state.) Therefore, we can obtain a restarting 1PFA with $4n{+}4$ states, solving $\texttt{TRIOS}(n, n^2)$ exactly. The expected number of the input tape sweeps can be bounded by

$$\varrho(n) = \textstyle\sum_{i=1}^{\infty} i{\cdot}\sigma(n){\cdot}(1{-}\sigma(n))^{i-1} \leq \sum_{i=1}^{\infty} i{\cdot}1{\cdot}\left(\frac{1}{e^n}\right)^{i-1} = \frac{1}{\left(1-\frac{1}{e^n}\right)^2} = \left(1{+}\tfrac{1}{e^n-1}\right)^2 \leq 1{+}o(1)\,.$$

Thus, for the input of length $|w|$, the expected running time is $(1{+}o(1)){\cdot}|w|$.

Finally, the family of promise problems $\texttt{TRIOS}(n, r)$, for $n, r \in \mathcal{N}_+$, does not give corresponding separations for two-way machines, since already 2DFAs can solve $\texttt{TRIOS}(n, r)$ with $O(n)$ states, for each r. This is done as follows. Let $\#x_iu_iv_i$, with $x_i, u_i, v_i \in \{0,1\}^n$, denote the i^{th} segment along the input. Processing of this segment begins with positioning the input head at the n^{th} bit of u_i, by moving $2n$ positions to the right from the symbol $\#$. Then, for $j = n, \ldots, 1$ (in that order), the machine verifies that $u_{i,j} = x_{i,j}$. (To move from the j^{th} bit of u_i to the j^{th} bit of x_i, just travel exactly n positions to the left; to move from the j^{th} bit of x_i to the $(j{-}1)^{\text{st}}$ bit of u_i, just travel exactly $n{-}1$ positions to the right. The machine detects that it has tested all n bits from the fact that, after traveling exactly n positions to the left for the j^{th} bit of x_i, it finds the symbol $\#$ at this position, instead of zero or one.) In a similar way, the

machine can also search for a bit position j satisfying $u_{i,j} < v_{i,j}$, this time iterating in ascending order, for $j = 1, \ldots, n$. If the i^{th} input segment passes the test successfully, the machine proceeds to examine the next segment $\#x_{i+1}u_{i+1}v_{i+1}$ or, for $i = r$, it accepts.

6 Two-Sided Bounded-Error Probabilistic Finite Automata

In Section 3, we have shown the limitations of Las Vegas PFAs. One-sided error[(ix)] PFAs have similar limitations since they can be simulated by nondeterministic or universal finite automata by removing the probabilities: If a promise problem $\mathtt{P} = (\mathtt{P}_{\mathtt{yes}}, \mathtt{P}_{\mathtt{no}})$ can be solved by a PFA rejecting each $w \in \mathtt{P}_{\mathtt{no}}$ with probability 1 (only members of $\mathtt{P}_{\mathtt{yes}}$ are accepted), then we obtain an NFA from the one-sided error PFA. Conversely, if each $w \in \mathtt{P}_{\mathtt{yes}}$ is accepted with probability 1 (only members of $\mathtt{P}_{\mathtt{no}}$ are rejected), then we obtain a finite automaton making only universal decisions (or an NFA for $\mathtt{P}_{\mathtt{no}}$) from the one-sided error PFA. In this section, we show that we have a different picture for PFAs with two-sided bounded-error.

First, we show that bounded-error 1PFAs can be very succinct compared to Las Vegas 2PFAs, 2AFAs, or any other simpler machines. In fact, we present a parallel result to that of Ambainis and Yakaryılmaz [2] but with bounded error instead of exact acceptance.

To this aim, consider $\mathcal{U}_p$, a 1PFA with the unary input alphabet $\Sigma = \{a\}$ and two states. Namely, let s_{I} be the initial and also the only accepting state, and s_{R} be the rejecting state. In the state s_{I}, the machine reads the symbol a staying in s_{I} with probability p and switching to s_{R} with the remaining probability $p' = 1-p$. In s_{R}, the machine just consumes the remaining part of the input, executing a single-state loop.

Now, let $\mathtt{UP}(p)$ be a promise problem such that

$$\begin{aligned} \mathtt{UP}_{\mathtt{yes}}(p) &= \{a^j \mid f_{\mathcal{U}_p}(a^j) \geq \tfrac{3}{4}\}\,, \\ \mathtt{UP}_{\mathtt{no}}(p) &= \{a^j \mid f_{\mathcal{U}_p}(a^j) \leq \tfrac{1}{4}\}\,. \end{aligned}$$

Clearly, for each $p \in (0,1)$, the promise problem $\mathtt{UP}(p)$ can be solved by a bounded-error one-way ε-free PFA with only two states, namely, by $\mathcal{U}_p$.

On the other hand, it is easy to show that the number of states required by any 2AFA for solving $\mathtt{UP}(p)$ (which covers all simpler models of automata as well) increases when p approaches 1: Since it is straightforward that the string a^j is accepted by $\mathcal{U}_p$ with probability p^j for each $j \geq 0$, we get $a^j \in \mathtt{UP}_{\mathtt{yes}}(p)$ only for finitely many values satisfying $j \leq \lfloor \log_p \frac{3}{4} \rfloor$, whereas $\mathtt{UP}_{\mathtt{no}}(p)$ contains infinitely many strings satisfying $j \geq \lceil \log_p \frac{1}{4} \rceil$. Let us denote these two critical lengths as

$$A_p = \lfloor \log_p \tfrac{3}{4} \rfloor \quad \text{and} \quad R_p = \lceil \log_p \tfrac{1}{4} \rceil\,.$$

Note also that $A_p \to +\infty$ as $p \to 1$.

Theorem 6.1 *For one-way/two-way deterministic/nondeterministic finite automata, and each $p \in (0,1)$, the number of states that is sufficient and necessary for solving the promise problem $\mathtt{UP}(p)$ is exactly A_p+1.*

(ix) We also cover the case of *unbounded-error*, i.e., the machines with no bound on the error and so the error can be arbitrarily close to 1 for some inputs.

Proof: First, we can easily design a 1DFA $\mathcal{D}$ (hence, any more powerful model as well) solving $\mathcal{U}_p$ with $A_p{+}1$ states. After the initial chain of $A_p{+}1$ accepting states, responsible to accept a^j for $j \in \{0, \ldots, A_p\}$, the automaton enters a rejecting state $s_{\rm R}$, in which it executes a single-step loop, consuming the rest of the input. By allowing undefined transitions in $\mathcal{D}$, the state $s_{\rm R}$ can be eliminated.

Second, before providing the matching lower bound, we need to present the so-called "$n \to n{+}n!$" method [40, 7, 14] (cf. also the proof of Theorem 4.1). To make the paper self-contained, we include also a proof, for a simplified version of this method that is sufficient for our purposes.

Claim. *Let $\mathcal{N}$ be a 2NFA (or any less powerful model) with n states and let a^m be a unary string, with $m \geq n$. Then, if $\mathcal{N}$ has a computation path traversing across a^m from left to right, starting in some state q_1 and ending in some q_2, it has also a computation path traversing across $a^{m+h\cdot m!}$, for each $h \geq 1$, starting and ending in the same states q_1 and q_2.*

The proof is based on the fact that $\mathcal{N}$ uses only n states, but the sequence of states along the left-to-right traversal across a^m is of length $m{+}1 > n$ (including q_1 at the beginning and q_2 at the end). Thus, in the course of this traversal, some state must be repeated. Hence, $\mathcal{N}$ executes a loop, by which it travels some $\ell \leq m$ positions to the right. Note that $m!/\ell$ is a positive integer. Hence, for each $h \geq 1$, by using $h{\cdot}m!/\ell$ additional iterations of this loop traveling ℓ positions, the machine traverses the string $a^{m+h\cdot m!}$, starting and ending in the same states q_1 and q_2. This completes the proof of the claim.

Now, let $\mathcal{N}$ be a 2NFA (or any less powerful model) solving $\texttt{UP}(p)$ with at most A_p states. Consider the inputs a^m and $a^{m+h\cdot m!}$, for $m = A_p$ and arbitrary $h \geq 1$. Clearly, $a^m = a^{A_p} \in \texttt{UP}_{\texttt{yes}}(p)$, and hence $\mathcal{N}$ has at least one accepting computation path for this input.

However, by the above claim, whenever this computation path traverses from the left endmarker of a^m to the opposite endmarker, starting in some state q_1 and ending in some q_2, an updated version of this path can traverse across $a^{m+h\cdot m!}$ starting and ending in the same states q_1 and q_2. By symmetry, the same holds for right-to-left traversals. Moreover, a U-turn starting and ending at the same endmarker of a^m can obviously do the same on the input $a^{m+h\cdot m!}$. Thus, for each $h \geq 1$, $\mathcal{N}$ has a computation path accepting $a^{m+h\cdot m!}$. But then $\mathcal{N}$ accepts infinitely many inputs, which contradicts the fact that all inputs longer than R_p belong to $\texttt{UP}_{\texttt{no}}(p)$ and should be rejected. Thus, $\mathcal{N}$ must use at least $A_p{+}1$ states. □

A lower bound for 2AFAs solving the promise problem $\texttt{UP}(p)$ can be obtained by combining Corollary 3.2, Theorem 6.1, and the result of Geffert and Okhotin [16], stating that an arbitrary n-state 2AFA can be converted into an equivalent 1NFA with $2^{O(n\cdot\log n)}$ states. From this one can derive that the number of states in any 2AFA solving $\texttt{UP}(p)$ is at least $\Omega(\frac{\log A_p}{\log\log A_p})$.

Corollary 6.2 *There exists $\{\texttt{UP}(p) \mid p \in (0,1)\}$, a family of unary promise problems such that bounded-error one-way ε-free* PFA*s with only two states are sufficient to solve all family, but the number of states required by* 2AFA*s (or by any simpler machines) to solve this family cannot be bounded by any constant.*

The error bound $\frac{1}{4}$ in the definition of $\texttt{UP}(p)$ can be replaced with an arbitrarily small error; the same results will follow by the use of the same reasoning.

Next, we present a separation result between bounded-error 1PFAs and 1DFAs (and so 2AFAs, or any other model capable of recognizing only regular languages). For this purpose, we use an idea given by Jibran and Yakaryılmaz [37]. It is known that 2PFAs can recognize some nonregular languages, e.g., $\texttt{EQ} = \{a^n b^n \mid n \in \mathcal{N}_+\}$ [12], with bounded error but this requires exponential time [11]. It was also

shown that EQ can be recognized by a restarting 1PFA for any error bound [42]. If the given string $a^m b^n$, where $m, n \in \mathcal{N}_+$, can be examined exponentially many times, then the algorithm given in [42] for EQ can distinguish between the cases of $m = n$ and $m \neq n$ with high probability. Based on this, we introduce a new family of promise problems $\texttt{ExpEQ}(c)$, for integer $c \geq 3$:

$$\begin{aligned} \texttt{ExpEQ}_{\texttt{yes}}(c) &= \{(a^m b^n)^{3\cdot(2c^2)^{m+n}\cdot\lceil \ln c\rceil} \mid m, n \in \mathcal{N}_+,\ m = n\}\,, \\ \texttt{ExpEQ}_{\texttt{no}}(c) &= \{(a^m b^n)^{3\cdot(2c^2)^{m+n}\cdot\lceil \ln c\rceil} \mid m, n \in \mathcal{N}_+,\ m \neq n\}\,. \end{aligned}$$

By [42], for each integer $c \geq 3$, there exists a restarting 1PFA $\mathcal{P}_c$ recognizing EQ such that, in one pass along each given input $a^m b^n$, the probability of giving an erroneous decision (rejecting if the input should be accepted or accepting if it should be rejected) is at least c times smaller than the probability of giving a correct decision. From this moment on, one pass of $\mathcal{P}_c$ along the given input $a^m b^n$ will be called a "round". To be more exact, let $\mathbb{A}$ and $\mathbb{R}$ denote, respectively, the probability that $\mathcal{P}_c$ accepts and rejects the input $a^m b^n$ in a single round. Then, by [42], the following holds:

$$\begin{aligned} \mathbb{R} &\leq \tfrac{1}{c}\cdot\mathbb{A}\,, \quad \text{if } m = n\,, \\ \mathbb{A} &\leq \tfrac{1}{c}\cdot\mathbb{R}\,, \quad \text{if } m \neq n\,. \end{aligned} \tag{1}$$

The remaining probability $\mathbb{N} = 1-\mathbb{A}-\mathbb{R}$ represents computations ending with a "don't know yet" result, after which $\mathcal{P}_c$ restarts another round. In addition, we know from [42] that

$$\mathbb{A} = \tfrac{1}{3\cdot(2c^2)^{m+n}}\,, \quad \text{independently of whether } m = n\,. \tag{2}$$

Note also that by combining (2) and (1) we get $\mathbb{A} > 0$ for $m = n$ and $\mathbb{R} > 0$ for $m \neq n$.

Finally, one of nice properties of these restarting 1PFAs is that they all use the same number of states, which is a constant that does not depend on c. (This is achieved by using arbitrarily small transition probabilities.)

Now we are ready to construct a 1PFA $\mathcal{P}'_c$ for solving $\texttt{ExpEQ}(c)$. For the given input $(a^m b^n)^t$, where

$$t = 3\cdot(2c^2)^{m+n}\cdot\lceil \ln c\rceil\,, \tag{3}$$

$\mathcal{P}'_c$ simulates $\mathcal{P}_c$ on $a^m b^n$ but, each time $\mathcal{P}_c$ restarts its computation from the very beginning of $a^m b^n$, the machine $\mathcal{P}'_c$ proceeds to the next copy of $a^m b^n$ along its own input. More precisely, if $\mathcal{P}_c$ gives the decision of "acceptance" in the course of one round, $\mathcal{P}'_c$ immediately accepts, by switching to an accepting state $s_{\rm A}$. Similarly, if $\mathcal{P}_c$ gives the decision of "rejection", $\mathcal{P}'_c$ immediately rejects, switching to a rejecting $s_{\rm R}$. (In both these states, $\mathcal{P}'_c$ consumes the rest of the input by executing single-state loops.) Otherwise, $\mathcal{P}'_c$ gets to the end of the current copy of $a^m b^n$ in a state $s_{\rm N}$, representing here a "don't know yet" outcome. That is, $\mathcal{P}'_c$ is ready to retry the entire procedure with the next copy of $a^m b^n$. Clearly, along the input $(a^m b^n)^t$, this can be repeated t times, after which $\mathcal{P}'_c$ halts in the state $s_{\rm N}$.

Since the number of states in $\mathcal{P}_c$ is a constant not depending on c, it is obvious that $\mathcal{P}'_c$ also uses a constant number of states that does not depend on c.

Now, let $\mathbb{A}_t$ and $\mathbb{R}_t$ denote the respective probabilities that $\mathcal{P}'_c$ accepts and rejects the input $(a^m b^n)^t$, by halting in the respective state $s_{\rm A}$ or $s_{\rm R}$. The remaining probability $\mathbb{N}_t = 1–\mathbb{A}_t–\mathbb{R}_t$ represents computations not deciding about acceptance or rejection, halting in the state $s_{\rm N}$.

Let us estimate $\mathbb{N}_t$ first. By combining $\mathbb{R} \geq 0$ with (2), (3), and the fact [23, Lm. A.3.60+] that $(1-\frac{1}{x})^x < \frac{1}{e}$ for each real $x > 1$ (in that order), we get, independently of whether $m = n$, that

$$\begin{aligned} \mathbb{N}_t &= \mathbb{N}^t = (1-\mathbb{A}-\mathbb{R})^t \leq (1-\mathbb{A})^t = \left(1-\tfrac{1}{3\cdot(2c^2)^{m+n}}\right)^t = \left(1-\tfrac{1}{3\cdot(2c^2)^{m+n}}\right)^{3\cdot(2c^2)^{m+n}\cdot\lceil \ln c\rceil} \\ &< \left(\tfrac{1}{e}\right)^{\lceil \ln c\rceil} \leq \tfrac{1}{c}\,. \end{aligned}$$

Therefore, using (1) and $\mathbb{A} > 0$, we can derive that the input $(a^m b^n)^t$ satisfying $m = n$ is accepted with probability at least

$$\begin{aligned} \mathbb{A}_t &= \textstyle\sum_{i=1}^{t} \mathbb{A}\cdot\mathbb{N}^{i-1} = \mathbb{A}\cdot\frac{1-\mathbb{N}^t}{1-\mathbb{N}} = \frac{\mathbb{A}}{\mathbb{A}+\mathbb{R}}\cdot(1-\mathbb{N}^t) \geq \frac{\mathbb{A}}{\mathbb{A}+\mathbb{A}/c}\cdot(1-\mathbb{N}^t) = \frac{c}{c+1}\cdot(1-\mathbb{N}^t) \\ &> \tfrac{c}{c+1}\cdot(1-\tfrac{1}{c}) = 1-\tfrac{2}{c+1}\,. \end{aligned}$$

Similarly, using (1) and $\mathbb{R} > 0$, the input $(a^m b^n)^t$ satisfying $m \neq n$ is rejected with probability at least

$$\begin{aligned} \mathbb{R}_t &= \textstyle\sum_{i=1}^{t} \mathbb{R}\cdot\mathbb{N}^{i-1} = \mathbb{R}\cdot\frac{1-\mathbb{N}^t}{1-\mathbb{N}} = \frac{\mathbb{R}}{\mathbb{A}+\mathbb{R}}\cdot(1-\mathbb{N}^t) \geq \frac{\mathbb{R}}{\mathbb{R}/c+\mathbb{R}}\cdot(1-\mathbb{N}^t) = \frac{c}{c+1}\cdot(1-\mathbb{N}^t) \\ &> \tfrac{c}{c+1}\cdot(1-\tfrac{1}{c}) = 1-\tfrac{2}{c+1}\,. \end{aligned}$$

Summing up, the success probability is always above $1-\frac{2}{c+1}$ and consequently the error probability is always below $\frac{2}{c+1}$. (It should be pointed out that the standard PFAs do not use "don't know" states. However, we can declare, by definition, the state s_{N} to be a rejecting state. This may potentially change all "don't know" answers to errors, rejecting inputs that should be accepted. However, this does not change the fact that $\mathbb{A}_t > 1-\frac{2}{c+1}$ for $m = n$, nor does it decrease $\mathbb{R}_t$ for $m \neq n$.)

By taking $c = \max\{3, \lceil \frac{2}{\varepsilon}\rceil - 1\}$ for arbitrarily small but fixed $\varepsilon > 0$, we obtain the error probability $\frac{2}{c+1} \leq \varepsilon$, keeping the same constant number of states for each ε. This gives:

Theorem 6.3 *For each fixed $\varepsilon > 0$, there exists a promise problem solvable by a* 1PFA *with bounded error ε, using a constant number of states that does not depend on ε, but there is no* 1DFA *(hence, no other machine capable of recognizing only regular languages) solving the same problem.*

Proof: We only need to show that $\texttt{ExpEQ}(c)$ cannot be solved by any 1DFA, for no $c \geq 3$. For contradiction, let $\mathcal{D}$ be a 1DFA solving $\texttt{ExpEQ}(c)$, for some $c \geq 3$. Without loss of generality, we assume that $\mathcal{D}$ does not have undefined transitions, and hence it always halts at the end of the input. (Otherwise, we can define all missing transitions by switching to a single new rejecting state, in which $\mathcal{D}$ scans the rest of the input.) Let $n \geq 1$ denote the number of states in $\mathcal{D}$.

Consider now the unary string a^n. Using the Claim presented in the proof of Theorem 6.1, we see that if $\mathcal{D}$ traverses across a^n, starting in some state q_1 and ending in some q_2, it will do the same also on the string $a^{n+h\cdot n!}$, for each $h \geq 1$. Clearly, the same holds for traversals of b^n and $b^{n+h\cdot n!}$.

Next, consider the input $(a^n b^n)^t$, where $t = 3\cdot(2c^2)^{2n+2n!}\cdot\lceil \ln c\rceil$. If, on this input, $\mathcal{D}$ halts in a state q', then, by the observation above, $\mathcal{D}$ must halt in the same state q' also on the inputs $(a^{n+n!}b^{n+n!})^t$ and $(a^n b^{n+2n!})^t$. Therefore, $\mathcal{D}$ accepts $(a^{n+n!}b^{n+n!})^t$ if and only if it accepts $(a^n b^{n+2n!})^t$. But this is a contradiction, since the former string should be accepted by $\mathcal{D}$ while the latter should be rejected. Consequently, there is no 1DFA solving $\texttt{ExpEQ}(c)$. □

7 Final Remarks

A thorough study of promise problems can reveal several interesting properties of computational models and give new fundamental insights about them. In automata theory, promise problems have mainly been used to show how quantum models can do much better than the classical ones when compared to the case of language recognition. In this paper, we initiated a systematic work on promise problems for classical one-way finite automata (deterministic, nondeterministic, alternating, and probabilistic). In this context, we have also shown that randomness can do much better than other classical resources.

Promise problems can further be investigated for different computational models and from different perspectives. Two-way finite state machines and counter or pushdown automata models are the first ones coming to the mind. Moreover, we believe that some long-standing open problems, formulated for language recognition, might be solved more easily in the case of promise problems.

Acknowledgements

The authors thank Beatrice Palano for providing us a copy of [4].

References

[1] Farid Ablayev, Aida Gainutdinova, Kamil Khadiev, and Abuzer Yakaryılmaz. Very narrow quantum OBDDs and width hierarchies for classical OBDDs. In *Descriptional Complexity of Formal Systems*, volume 8614 of *Lecture Notes in Computer Science*, pages 53–64. Springer-Verlag, 2014.

[2] Andris Ambainis and Abuzer Yakaryılmaz. Superiority of exact quantum automata for promise problems. *Information Processing Letters*, 112(7):289–291, 2012.

[3] Zuzana Bednárová, Viliam Geffert, Klaus Reinhardt, and Abuzer Yakaryılmaz. New results on the minimum amount of useful space. *Computing Research Repository*, abs/1405.2892v2, 2015.

[4] Maria Paola Bianchi, Carlo Mereghetti, and Beatrice Palano. Complexity of promise problems on classical and quantum automata. In *Gruska Festschrift*, volume 8808 of *Lecture Notes in Computer Science*, pages 161–175. Springer-Verlag, 2014.

[5] R. G. Bukharaev. Probabilistic automata. *Journal of Mathematical Sciences*, 13(3):359–386, 1980.

[6] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[7] Richard Chang, Juris Hartmanis, and Desh Ranjan. Space bounded computations: Review and new separation results. *Theoretical Computer Science*, 80:289–302, 1991.

[8] Anne Condon and Richard J. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *Proc. Foundations of Computer Science*, pages 462–467, 1989.

[9] Pavol Ďuriš, Juraj Hromkovič, José D. P. Rolim, and Georg Schnitger. Las Vegas versus determinism for one-way communication complexity, finite automata, and polynomial-time computations. In *Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *Lecture Notes in Computer Science*, pages 117–128. Springer-Verlag, 1997.

[10] Cynthia Dwork and Larry Stockmeyer. Finite state verifiers I: The power of interaction. *Journal of the ACM*, 39(4):800–828, 1992.

[11] Cynthia Dwork and Larry J. Stockmeyer. A time complexity gap for two-way probabilistic finite-state automata. *SIAM Journal on Computing*, 19(6):1011–1123, 1990.

[12] Rūsiņš Freivalds. Probabilistic two-way machines. In *Mathematical Foundations of Computer Science*, volume 118 of *Lecture Notes in Computer Science*, pages 33–45. Springer-Verlag, 1981.

[13] Aida Gainutdinova and Abuzer Yakaryılmaz. Unary probabilistic and quantum automata on promise problems. In *Developments in Language Theory*, volume 9168 of *Lecture Notes in Computer Science*, pages 252–263. Springer-Verlag, 2015.

[14] Viliam Geffert. Nondeterministic computations in sublogarithmic space and space constructibility. *SIAM Journal on Computing*, 20:484–498, 1991.

[15] Viliam Geffert. An alternating hierarchy for finite automata. *Theoretical Computer Science*, 445:1–24, 2012.

[16] Viliam Geffert and Alexander Okhotin. Transforming two-way alternating finite automata to one-way nondeterministic automata. In *Mathematical Foundations of Computer Science, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 291–302. Springer-Verlag, 2014.

[17] Viliam Geffert and Abuzer Yakaryılmaz. Classical automata on promise problems. In *Descriptional Complexity of Formal Systems*, volume 8614 of *Lecture Notes in Computer Science*, pages 126–137. Springer-Verlag, 2014.

[18] Oded Goldreich. On promise problems: A survey. In *Essays in Memory of Shimon Even*, volume 3895 of *Lecture Notes in Computer Science*, pages 254–290. Springer-Verlag, 2006.

[19] Jozef Gruska, Lvzhou Li, and Shenggen Zheng. Recognizability versus solvability of promise problems in finite classical and quantum automata framework. *Computing Research Repository*, abs/1411.3870v2, 2014.

[20] Jozef Gruska, Daowen Qiu, and Shenggen Zheng. Generalizations of the distributed Deutsch-Jozsa promise problem. *Mathematical Structures in Computer Science* (to appear), doi:10.1017/S0960129515000158, 2015. Also arXiv:1402.7254.

[21] Jozef Gruska, Daowen Qiu, and Shenggen Zheng. Potential of quantum finite automata with exact acceptance. *International Journal of Foundations of Computer Science*, 36(3):381–398, 2015.

[22] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Prentice Hall, 2007.

[23] Juraj Hromkovič. *Design and Analysis of Randomized Algorithms (Introduction to Design Paradigms)*. Springer-Verlag, 2005.

[24] Juraj Hromkovič and Georg Schnitger. On the power of Las Vegas for one-way communication complexity, OBDDs, and finite automata. *Information and Computation*, 169(2):284–296, 2001.

[25] Galina Jirásková and Giovanni Pighizzini. Optimal simulation of self-verifying automata by deterministic automata. *Information and Computation*, 209:528–535, 2011.

[26] Christos A. Kapoutsis. Removing bidirectionality from nondeterministic finite automata. In *Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 544–555. Springer-Verlag, 2005.

[27] Christos A. Kapoutsis, Richard Královič, and Tobias Mömke. Size complexity of rotating and sweeping automata. *Journal of Computer and System Sciences*, 78(2):537–558, 2012.

[28] Yu. I. Kuklin. Two-way probabilistic automata. *Avtomatika i vyčistiteľnaja tekhnika*, 5:35–36, 1973. (Russian).

[29] Rajendra Kumar. *Theory of Automata, Languages, and Computation*. Tata McGraw-Hill, 2010.

[30] Richard E. Ladner, Richard J. Lipton, and Larry J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal on Computing*, 13(1):135–155, 1984.

[31] Carlo Mereghetti and Giovanni Pighizzini. Optimal simulations between unary automata. *SIAM Journal on Computing*, 30:1976–1992, 2001.

[32] Yumiko Murakami, Masaki Nakanishi, Shigeru Yamashita, and Katsumasa Watanabe. Quantum versus classical pushdown automata in exact computation. *IPSJ Digital Courier*, 1:426–435, 2005.

[33] Masaki Nakanishi. Quantum pushdown automata with a garbage tape. In *SOFSEM: Theory and Practice of Computer Science*, volume 8939 of *Lecture Notes in Computer Science*, pages 352–363. Springer-Verlag, 2015.

[34] Masaki Nakanishi and Abuzer Yakaryılmaz. Classical and quantum counter automata on promise problems. In *Conference on Implementation and Application of Automata*, volume 9223 of *Lecture Notes in Computer Science*, pages 224–237. Springer-Verlag, 2015.

[35] Azaria Paz. *Introduction to Probabilistic Automata*. Academic Press, New York, 1971.

[36] Michael O. Rabin. Probabilistic automata. *Information and Control*, 6:230–243, 1963.

[37] Jibran Rashid and Abuzer Yakaryılmaz. Implications of quantum automata for contextuality. In *Conference on Implementation and Application of Automata*, volume 8587 of *Lecture Notes in Computer Science*, pages 318–331. Springer-Verlag, 2014. arXiv:1404.2761.

[38] Klaus Reinhardt and Abuzer Yakaryılmaz. The minimum amount of useful space: New results and new directions. In *Developments in Language Theory*, volume 8633 of *Lecture Notes in Computer Science*, pages 315–326. Springer-Verlag, 2014.

[39] Michael Sipser. Lower bounds on the size of sweeping automata. *Journal of Computer and System Sciences*, 21(2):195–202, 1980.

[40] Richard Edwin Stearns, Juris Hartmanis, and Philip M. Lewis II. Hierarchies of memory limited computations. In *IEEE Conference Record on Switching Circuit Theory and Logical Design*, pages 179–190, 1965.

[41] John Watrous. Quantum computational complexity. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 7174–7201. Springer-Verlag, 2009.

[42] Abuzer Yakaryılmaz and A. C. Cem Say. Succinctness of two-way probabilistic and quantum finite automata. *Discrete Mathematics and Theoretical Computer Science*, 12(2):19–40, 2010.

[43] Shenggen Zheng, Jozef Gruska, and Daowen Qiu. On the state complexity of semi-quantum finite automata. In *Language and Automata Theory and Applications*, volume 8370 of *Lecture Notes in Computer Science*, pages 601–612. Springer-Verlag, 2014.

[44] Shenggen Zheng, Daowen Qiu, Jozef Gruska, Lvzhou Li, and Paulo Mateus. State succinctness of two-way finite automata with quantum and classical states. *Theorerical Computer Science*, 499:98–112, 2013.