Exact Affine Counter Automata[*]

Masaki Nakanishi
Department of Education, Art and Science, Yamagata University,
Yamagata, 990–8560, Japan
masaki@cs.e.yamagata-u.ac.jp

Kamil Khadiev
University of Latvia, Faculty of Computing, Center for Quantum Computer Science, Rīga, Latvia
Kazan Federal University, Institute of Computational Mathematics and IT,
Kremlevskaya str. 18, Kazan, 420008, Russia
kamilhadi@gmail.com

Krišjānis Prūsis Jevgēnijs Vihrovs Abuzer Yakaryılmaz
University of Latvia, Faculty of Computing, Center for Quantum Computer Science, Rīga, Latvia
krisjanis.prusis@lu.lv, jevgenijs.vihrovs@lu.lv, abuzer@lu.lv

We introduce an affine generalization of counter automata, and analyze their ability as well as affine finite automata. Our contributions are as follows. We show that there is a language that can be recognized by exact realtime affine counter automata but by neither 1-way deterministic pushdown automata nor realtime deterministic k-counter automata. We also show that a certain promise problem, which is conjectured not to be solved by two-way quantum finite automata in polynomial time, can be solved by Las Vegas affine finite automata. Lastly, we show that how a counter helps for affine finite automata by showing that the language MANYTWINS, which is conjectured not to be recognized by affine, quantum or classical finite state models in polynomial time, can be recognized by affine counter automata with one-sided bounded-error in realtime.

1 Introduction

Quantum computation models can be more powerful than their classical counterparts. This is mainly because quantum models are allowed to use negative amplitudes, by which interference can occur between configurations. In order to mimic quantum interference classically, recently a new concept called affine computation was introduced [4] and its finite automata versions (AfAs) have been examined [4, 15, 2, 8]. Some underlying results are as follows: (i) they are more powerful than their probabilistic and quantum counterparts (PFAs and QFAs) with bounded and unbounded error; (ii) one-sided bounded-error AfAs and nondeterministic QFAs define the same class when using rational number transitions; and, (iii) AfAs can distinguish any given pair of strings by using two states with zero-error. Very recently, affine OBDD was introduced in [9] and it was shown that they can be exponentially narrower than bounded-error quantum and classical OBDDs.

In this paper, we introduce (realtime) AfA augmented with a counter (AfCAs), and analyze their ability as well as Las Vegas AfAs. It is already known that AfAs can simulate QFAs exactly by a quadratic increase in the number of states [15]. However, this simulation cannot be extended to the simulation of QFAs with a counter (QCAs). Therefore, the quantum interference used by QCAs cannot be trivially used by AfCAs. Besides, the well-formed conditions for QCAs can be complicated, but as

[*]Parts of the research work were done while Yakaryılmaz was visiting Yamagata University in November 2016 and all authors were visiting Kyoto University in March 2017.

E. Csuhaj-Varjú, P. Dömösi, Gy. Vaszil (Eds.): 15th International Conference on Automata and Formal Languages (AFL 2017)
EPTCS 252, 2017, pp. 205–218, doi:10.4204/EPTCS.252.20

seen soon, they are easy to check for AfCAs. Thus, we believe that AfCAs may present classical and simpler setups for the tasks done by QCAs.

Our main contribution in this paper is that we show that there is a language that can be recognized exactly (zero-error) by realtime AfCAs but neither by 1-way deterministic pushdown automata nor by realtime deterministic k-counter automata. This is the first separation result concerning AfCAs. This is a strong result since an exact one-way probabilistic one-counter automaton (PCA) is simply a one-way deterministic one-counter automaton (DCA) and it is still open whether exact one-way QCAs are more powerful than one-way DCAs and whether bounded-error one-way QCAs are more powerful than one-way bounded-error PCAs (see [14, 10] for some affirmative results).

In [11], it was shown that a certain promise problem can be solved by two-way QFAs (2QCFAs) exactly but in exponential time, and bounded-error two-way PFAs (2PFAs) can solve the problem only if they are allowed to use logarithmic amount of memory. We show that the same problem can be solved by realtime Las Vegas AfAs or AfAs with restart in linear expected time. Lastly, we address the language `MANYTWINS`, which is conjectured not to be recognized by affine, quantum or classical finite state models in polynomial time. We show how a counter helps for AfAs by showing that `MANYTWINS` can be recognized by AfCAs with one-sided bounded-error in realtime read mode.

In the next section, we provide the necessary background. Our main results are given in Sections 3, 4 and 5, respectively. Section 6 concludes the paper.

2 Background

We assume the reader to have the knowledge of automata theory, and familiarity with the basics of probabilistic and quantum automata. We refer [13] and [1] for the quantum models.

Throughout the paper, the input alphabet is denoted as Σ not including the left end-marker (¢) and the right end-marker (\$). The set $\widetilde{\Sigma}$ denotes $\Sigma \cup \{¢, \$\}$. For a given input $w \in \Sigma^*$, $|w|$ is the length of w, $w[i]$ is the i-th symbol of w, and $\tilde{w} = ¢w\$$. For any given string $w \in \{1,2\}^*$, $e(w)$ denotes the encoding of w in base-3. The value $\overline{1}$ in a vector represents the value that makes the vector summation equal to 1, i.e., if the summation of all other entries is x, then it represents the value $1-x$.

A (realtime) affine finite automaton (AfA) [4] A is a 5-tuple

$$A = (S, \Sigma, \{M_\sigma \mid \sigma \in \widetilde{\Sigma}\}, s_I, S_a),$$

where $S = \{s_1, \ldots, s_n\}$ is a finite set of states, Σ is a finite set of input symbols, M_σ is the $n \times n$ affine transition matrix for symbol $\sigma \in \widetilde{\Sigma}$, $s_I \in S$ is the initial state, and $S_a \subseteq S$ is a finite set of accepting states.

We consider a one-to-one correspondence between the set of configurations (i.e., the set of states S) and the standard basis of an n-dimensional real vector space. Then, any affine state is represented as an n-dimensional real vector such that the summation of all entries is equal to 1. For a given input $w \in \Sigma^*$, A starts its computation in the initial affine state v_0, where the I-th entry is 1 and the others are zeros. Then, it reads $\tilde{w}$ symbol by symbol from the left to the right and for each symbol the affine state is changed as follows:

$$v_j = M_{\tilde{w}[j]} v_{j-1},$$

where $1 \leq j \leq |\tilde{w}|$. To be a well-formed machine, the summation of entries of each column of M_σ must be 1. The final state is denoted as $v_f = v_{|\tilde{w}|}$. At the end, the weighting operator[1] returns the probability

[1]This operator returns the weight of each value in the l_1 norm of the vector.

of observing each state as

$$Pr[\text{observing } s_i] = \frac{|v_f[i]|}{|v_f|},$$

where $1 \leq i \leq |S|$, $v_f[i]$ is the i-th entry of v_f, and $|v_f|$ is l_1 norm of v_f. Thus, the input w is accepted by A with probability

$$f_A(w) = \sum_{s_i \in S_a} \frac{|v_f[i]|}{|v_f|}.$$

Next, we define a new affine model. A (realtime) affine counter automaton (AfCA) A is an AfA augmented with a counter. Formally, it is a 5-tuple

$$A = (S, \Sigma, \delta, s_I, S_a),$$

where the difference from an AfA

$$\delta : S \times \widetilde{\Sigma} \times \{Z, NZ\} \times S \times \{-1, 0, +1\} \to \mathbb{R}$$

is the transition function governing the behavior of A such that when it is in the state $s \in S$, reads the symbol $\sigma \in \widetilde{\Sigma}$, and the current status of the counter is $\theta \in \{Z, NZ\}$ (Z : zero, NZ : nonzero), it makes the following transition with value $\delta(s, \sigma, \theta, s', d)$: it switches to the state $s' \in S$ and updates the value of the counter by $d \in \{-1, 0, +1\}$. To be a well-formed affine machine, the transition function must satisfy that, for each triple $(s, \sigma, \theta) \in S \times \widetilde{\Sigma} \times \{Z, NZ\}$,

$$\sum_{s' \in S, d \in \{-1,0,+1\}} \delta(s, \sigma, \theta, s', d) = 1.$$

Remark that the value of the counter can be updated by a value in $\{-t, \ldots, +t\}$ for some $t > 1$ but this does not change the computational power of the model (see [18] for more details).

Any classical configuration of A is formed by a pair $(s, c) \in S \times \mathbb{Z}$, where s is the deterministic state and c is the value of the counter. Let $w \in \Sigma^*$ be the given input and $m = |¢w\$|$. Since all possible values of the counter are in $\{-m, \ldots, m\}$, the total number of classical configurations is $N = m|S|$. We denote the set $\{(s, c) \mid s \in S, c \in \{-m, \ldots, m\}\}$ for w as $\mathscr{C}^w$. In a similar way to AfAs, the automaton A reads $¢w\$$ symbol by symbol from the left to the right and A operates on the classical configurations. Each such configuration, say (s, c), can be seen as the state of an affine system which we represent as $\langle s, c \rangle$ (a vector in the standard basis of $\mathbb{R}^N$). During the computation, similarly to quantum models, A can be in more than one classical configuration with some values, i.e.

$$v = \sum_{(s,c) \in \mathscr{C}^w} \alpha_{s,c} \langle s, c \rangle \text{ satisfying that } \sum_{(s,c) \in \mathscr{C}^w} \alpha_{s,c} = 1.$$

Due to their simplicity, we use such linear combinations to trace the computation of an AfCA. Then we can also define the affine transition matrix M_σ for each $\sigma \in \tilde{\Sigma}$ as follows:

$$M_\sigma \langle s, c \rangle = \sum_{s' \in S, d \in \{-1,0,+1\}} \delta(s, \sigma, \theta(c), s', d) \langle s', c + d \rangle,$$

where $\theta(c) = Z$ if $c = 0$, and $\theta(c) = NZ$ otherwise. At the beginning of computation, A is in $v_0 = \langle s_I, 0 \rangle$. Then, after reading each symbol, the affine state of the machine is updated, i.e.

$$v_0 \to v_1 \to \cdots \to v_m, \text{ where } v_{i+1} = M_{\tilde{w}[i+1]} v_i \ (0 \leq i \leq m-1).$$

After reading the whole input, the final affine state becomes

$$v_f = v_m = \sum_{(s,c)\in\mathscr{C}^w} \beta_{s,c}\langle s,c\rangle,$$

and then the weighting operator is applied and the input is accepted with probability

$$f_A(w) = \sum_{s\in S_a, c\in\{-m,\ldots,m\}} \frac{|\beta_{s,c}|}{|v_f|},$$

which is the total weight of "accepting" configurations out of all configurations at the end.

We can extend AfCAs to have multiple counters (affine k-counter automata (AfkCAs)) in a straightforward way; the transition function is extended to $\delta : S\times\tilde{\Sigma}\times\{Z,NZ\}^k\times S\times\{-1,0,+1\}^k \longrightarrow \mathbb{R}$.

A (realtime) Las Vegas automaton is obtained from a standard one by splitting the set of states into three: the set of accepting, rejecting, and neutral states. When it enters one of them at the end of the computation, then the answers of "accepting", "rejecting", and "don't know" are given, respectively.

A (realtime) automaton with restart [19] is similar to a Las Vegas automaton, the set of states of it is split into "accepting", "rejecting", and "restarting" states. At the end of the computation, if the automaton enters a restarting state, then all the computation is restarted from the beginning. An automaton with restart can be seen as a restricted sweeping two-way automaton. The overall accepting probability can be simply obtained by making a normalization over the accepting and rejecting probabilities in a single round (see also [20]).

If an affine automaton is restricted to use only non-negative values as an entry of its transition matrix, then it becomes a probabilistic automaton. As a further restriction, if only 1 and 0 are allowed to be used, then it becomes a deterministic automaton. Thus, any (realtime) AfCA using only 0 and 1 as transition values is a (realtime) deterministic counter automaton (realtime DCA).

All the models mentioned above are realtime models, whose tape head moves to the right at each step. Next, we introduce a one-way model, whose tape head is allowed to move to the right or stay at the same position, but not allowed to move to the left.

A one-way deterministic pushdown automaton (1DPA) A is a 7-tuple

$$A = (S,\Sigma,\Gamma,\delta,s_I,Z_0,S_a),$$

where $S=\{s_1,\ldots,s_n\}$ is a finite set of states, Σ is a finite set of input symbols, Γ is a finite set of stack symbols, $\delta : S\times\tilde{\Sigma}\times\Gamma\times S\times\Gamma^*\times\{0,1\}\longrightarrow\{0,1\}$ is a transition function, s_I is the initial state, Z_0 is the initial stack symbol, and $S_a\subseteq S$ is the set of accepting states. To be a well-formed machine, the transition function must satisfy that, for each triple (s,σ,γ),

$$\sum_{s'\in S,\gamma'\in\Gamma^*,D\in\{0,1\}} \delta(s,\sigma,\gamma,s',\gamma',D) = 1.$$

For a given input ¢w\$, the automaton A starts its computation with the following initial configuration:

- the initial state is s_I,
- the stack has only the initial stack symbol Z_0,
- the tape head points to the left endmarker.

Then, at each step of the computation, A is updated according to the transition function δ, i.e., $\delta(s,\sigma,\gamma,s',\bar{\gamma}',D) = 1$ implies that if the current state is s, the scanned input symbol is σ and the stack-top symbol is γ, then it moves to the state s' and updates the stack by deleting the stack-top symbol and pushing $\bar{\gamma}'$. Also, the tape head moves to D where $D = 0$ means "stationary" and $D = 1$ means "move to the right". Note that a move with $D = 0$ is called an ε-move. For an input word w, if A reaches an accepting state, then w is accepted.

A promise problem $\mathtt{P} \subseteq \Sigma^*$ is formed by two disjoint subsets: yes-instances $\mathtt{P_{yes}}$ and no-instances $\mathtt{P_{no}}$. An automaton A solves P with error bound $\varepsilon < \frac{1}{2}$ if each yes-instance (resp. no-instance) is accepted (resp. rejected) with probability at least $1-\varepsilon$. If all yes-instances (resp. no-instances) are accepted with probability 1 (resp. 0), then the error bound is called one-sided. If $\varepsilon = 0$, then the problem is said to be solved exactly (or with zero-error). A promise problem is solved by a Las Vegas algorithm with success probability $p < 1$ if any yes-instance (resp. no-instance) is accepted (resp. rejected) with probability $p' \geq p$ and the answer of "don't know" is given with the remaining probability $1 - p'$. If $\mathtt{P_{yes}} \cup \mathtt{P_{no}} = \Sigma^*$, then it is called language recognition (for $\mathtt{P_{yes}}$) instead of solving a promise problem.

3 Exact separation

We start with defining a new language END:

$$\mathtt{END} = \{w \in \{0,1,2\}^*2\{0,1,2\}^* \mid w^r[|w|_2] = 1\},$$

where $|w|_2$ is the number of symbols 2 in w, w^r is the reverse of w, and $w^r[|w|_2]$ is the $(|w|_2)$-th symbol of w^r.

Theorem 1. *The language* END *is recognized by an AfCA A exactly.*

Proof. We will use two states (s_1,s_2) for deterministic computation and five states (p_0,p_1,p_2,p_3,p_4) for affine computation. The initial states are s_1 and p_0. In other words, we consider a product of a 2-state deterministic finite automaton and a 5-state affine counter automaton.

The classical part is responsible for checking whether the given input has at least one symbol 2. For this purpose, s_1 switches to s_2 after reading a symbol 2 and then never leaves s_2 until the end of the computation. If the automaton ends in state s_1, the input is rejected.

From now on, we focus on the affine transitions. The computation starts in the following affine configuration:

$$\langle p_0, 0\rangle,$$

where p_0 is the affine state and 0 represents the counter value.

After reading the left end-marker, p_0 goes to p_0, p_1, and p_2 with the values 1, 1, and -1, respectively, without changing the counter value. Then, the affine state becomes

$$\langle p_0,0\rangle + \langle p_1,0\rangle - \langle p_2,0\rangle.$$

We list the all transitions until reading the right end-marker below, in which c can be any integer representing the counter value. Remark that the counter status is never checked in these transitions.

- When reading a symbol 0:
  - $\langle p_0,c\rangle \to \langle p_0,c\rangle$
  - $\langle p_1,c\rangle \to \langle p_1,c\rangle - \frac{1}{2}\langle p_3,c+1\rangle + \frac{1}{2}\langle p_4,c+1\rangle$

- $\langle p_2, c\rangle \to \langle p_2, c\rangle$
- $\langle p_3, c\rangle \to \langle p_3, c+1\rangle$
- $\langle p_4, c\rangle \to \langle p_4, c+1\rangle$

• When reading a symbol 1:

- $\langle p_0, c\rangle \to \langle p_0, c\rangle$
- $\langle p_1, c\rangle \to \langle p_1, c\rangle + \frac{1}{2}\langle p_3, c+1\rangle - \frac{1}{2}\langle p_4, c+1\rangle$
- $\langle p_2, c\rangle \to \langle p_2, c\rangle$
- $\langle p_3, c\rangle \to \langle p_3, c+1\rangle$
- $\langle p_4, c\rangle \to \langle p_4, c+1\rangle$

• When reading a symbol 2:

- $\langle p_0, c\rangle \to \langle p_0, c\rangle$
- $\langle p_1, c\rangle \to \langle p_1, c-1\rangle - \frac{1}{2}\langle p_3, c\rangle + \frac{1}{2}\langle p_4, c\rangle$
- $\langle p_2, c\rangle \to \langle p_2, c-1\rangle$
- $\langle p_3, c\rangle \to \langle p_3, c\rangle$
- $\langle p_4, c\rangle \to \langle p_4, c\rangle$

Let w be the input, let $n = |w|$, $x = w^r$, and $|w|_2 = k \geq 1$. The affine state before reading the right end-marker is

$$\langle p_0, 0\rangle + \langle p_1, -k\rangle - \langle p_2, -k\rangle + \sum_{i=1}^{|x|=n} (-1)^{x[i]} \left(-\frac{1}{2}\langle p_3, i-k\rangle + \frac{1}{2}\langle p_4, i-k\rangle \right).$$

Here the first three terms are trivial since (i) $\langle p_0, 0\rangle$ never leaves itself, and, (ii) the values of p_1 and p_2 are never changed and the counter value is decreased for each symbol 2 (k times in total).

In order to verify the last term, we closely look into the step when reading an arbitrary input symbol, say $w[j]$ ($1 \leq j \leq n$). Suppose that $t \in \{0, \dots, j-1\}$ symbols 2 have been read until now. We calculate the final counter values of the configurations with states p_3 and p_4 that are created from $\langle p_1, -t\rangle$ in this step.

- If $w[j]$ is 0 or 1, then the following configurations are created:

  $$-\frac{1}{2}\langle p_3, -t+1\rangle + \frac{1}{2}\langle p_4, -t+1\rangle \text{ or } \frac{1}{2}\langle p_3, -t+1\rangle - \frac{1}{2}\langle p_4, -t+1\rangle,$$

  respectively. In the remaining part of the computation, $k-t$ symbols 2 and $n-j-(k-t)$ symbols 0 or 1 are read. When reading a symbol 2, the counter value remains the same and it is increased by 1 when a symbol 0 or 1 is read. Thus, their final counter values hit $-t+1+n-j-(k-t) = (n-j+1)-k$.

- If $w[j]$ is 2, then the following configuration is created:

  $$-\frac{1}{2}\langle p_3, -t\rangle + \frac{1}{2}\langle p_4, -t\rangle.$$

  In the remaining part of the computation, $k-t-1$ symbols 2 and $n-j-(k-t-1)$ symbols 0 or 1 are read. Then, as explained in the previous item, their final counter values hit $-t+n-j-(k-t-1) = (n-j+1)-k$.

It is clear that $n-j+1$ refers to i in the above equation, i.e. $x[i]=x[n-j+1]=w[j]$, and so the correctness of this equation is verified. Moreover, we can follow that the counter value for these configurations is zero if and only if $i=n-j+1=k$, and, this refers to the input symbol $x[|w|_2]=(w^r)[|w|_2]$.

Therefore, if we can determine the value of $w^r[|w|_2]$ with zero error, then we can also determine whether the given input is in the language or not with zero error. For this purpose, we use the following transitions on the right end-marker in which the value of the counter is not changed:

- Both states p_1 and p_2 switch to p_1. Thus, the pair $\langle p_1,-k\rangle - \langle p_2,-k\rangle$ disappears.
- If the value of the counter is non-zero, both states p_3 and p_4 switch to p_3. Thus, each pair of the form $(-1)^{x[i]}\left(\frac{1}{2}\langle p_1,\neq 0\rangle - \frac{1}{2}\langle p_2,\neq 0\rangle\right)$ disappears ($i\neq k$).
- If the value of the counter is zero, both states p_3 and p_4 switch to themselves. Moreover, the state p_0 switches to p_3 and p_4 with values of $\frac{1}{2}$. Then, the following interference appears:

$$v_f=(-1)^{x[i]}\left(-\frac{1}{2}\langle p_3,0\rangle+\frac{1}{2}\langle p_4,0\rangle\right)+\frac{1}{2}\langle p_3,0\rangle+\frac{1}{2}\langle p_4,0\rangle.$$

If $x[i]=1$, then $v_f=\langle p_3,0\rangle$. If $x[i]=0$ or $x[i]=2$, then $v_f=\langle p_4,0\rangle$. Thus, by setting (s_2,p_3) as the only accepting state, we can obtain the desired machine. $\square$

Next, we prove that the language END is recognized neither by 1-way deterministic pushdown automata (1DPAs) nor by realtime deterministic k-counter automata (realtime DkCAs)[2]. For this purpose, we introduce the following lemma (the pumping lemma for deterministic context-free languages (DCFLs)) [21].

Lemma 1. *(Pumping Lemma for DCFLs [21]) Let L be a DCFL. Then there exists a constant C for L such that for any pair of words w, $w'\in L$ if*

(1) $w=xy$ and $w'=xz$, $|x|>C$, and

(2) ${}^{(1)}y={}^{(1)}z$, where ${}^{(1)}w$ is defined to be the first symbol of w

$${}^{(1)}w=\begin{cases} x & \text{if } |w|>1, w=xy, \text{ and } |x|=1;\\ w & \text{if } |w|\le 1,\end{cases}$$

then either (3) or (4) is true:

(3) there is a factorization $x=x_1x_2x_3x_4x_5$, $|x_2x_4|\ge 1$ and $|x_2x_3x_4|\le C$, such that for all $i\ge 0$ $x_1x_2^ix_3x_4^ix_5y$ and $x_1x_2^ix_3x_4^ix_5z$ are in L;

(4) there exist factorizations $x=x_1x_2x_3$, $y=y_1y_2y_3$ and $z=z_1z_2z_3$, $|x_2|\ge 1$ and $|x_2x_3|\le C$, such that for all $i\ge 0$ $x_1x_2^ix_3y_1y_2^iy_3$ and $x_1x_2^ix_3z_1z_2^iz_3$ are in L. $\square$

Theorem 2. *The language* END *cannot be recognized by any 1DPA.*

Proof. We assume that END is a DCFL and let C be the constant for END in Lemma 1. Choose $w=2^p10^{p-1}\in$ END and $w'=2^p10^{p-1}10^{p-1}$ for some integer $p>C+1$, and set $x=2^p10^{p-2}, y=0$ and $z=010^{p-1}$. Then, $w=xy$ and $w'=xz$ satisfy (1) and (2) in Lemma 1.

We first consider the case that (3) holds. In order to satisfy $x_1x_2^ix_3x_4^ix_5y\in$ END for $i\ge 0$, x_2x_4 must not have the symbol 1 (otherwise, $x_1x_2^0x_3x_4^0x_5y\notin$ END). Thus, x_2 and x_4 are of the form 2^t or 0^t for some

[2]Since 1-way (with epsilon moves) deterministic 2-counter automata can simulate Turing machines, the restriction of "realtime" is essential.

constant t. If $x_2 = 2^{t_1}$ and $x_4 = 2^{t_2}$, $x_1x_2^ix_3x_4^ix_5y \notin \texttt{END}$ for $i \neq 1$. Similarly, $x_2 = 0^{t_1}$ and $x_4 = 0^{t_2}$ cannot occur. Thus, the only possible choice is $x_2 = 2^{t_1}$ and $x_4 = 0^{t_2}$ for some t_1 and t_2. In order to satisfy $x_1x_2^ix_3x_4^ix_5y \in \texttt{END}$ for $i \geq 0$, $t_1 = t_2$ must hold. However, This causes $x_1x_2^ix_3x_4^ix_5z \notin \texttt{END}$ for $i \neq 1$. Thus, (3) does not hold.

Next, we consider the case that (4) holds. Since $|x_2x_3| \leq C$, x_2 can have only 0s. Thus, for any factorization $w = x_1x_2x_3y_1y_2y_3$, $x_1x_2^ix_3y_1y_2^iy_3 \notin \texttt{END}$ for $i \neq 1$. Thus, (4) does not hold. This is a contradiction. Therefore, END is not a DCFL, which implies no 1DPA can recognize END. □

Theorem 3. *The language* END *cannot be recognized by any realtime DkCA.*

Proof. We assume that there exists a realtime DkCA that recognizes END. We consider an input of the form $w = 2^mxy (x \in \{0,1\}^m, y \in \{0,1\}^*)$. Then we have 2^m possible x's. For any x_1 and $x_2 \in \{0,1\}^m (x_1 \neq x_2)$, we will show that there exists a y such that $2^mx_1y \in \texttt{END}$ and $2^mx_2y \notin \texttt{END}$ or vice versa.

We assume that $x_1[i] \neq x_2[i]$. Note that there exists such an i since $x_1 \neq x_2$. We also assume that $x_1[i] = 1$ and $x_2[i] = 0$ without loss of generality. We set $y = 0^{i-1}$. Then $(2^mx_1y)^R[m] = x_1[i] = 1$ and $(2^mx_2y)^R[m] = x_2[i] = 0$. Thus, $2^mx_1y \in \texttt{END}$ and $2^mx_2y \notin \texttt{END}$. Therefore, the configurations after reading 2^mx_1 and 2^mx_2 must be different. However, the number of possible configurations for a realtime DkCA after reading the partial input 2^mx is $O(m)$ while there are 2^m possible x's. This is a contradiction. □

Currently, we do not know any QCA algorithm solving END. Moreover, recently another promise problem solvable by exact QCAs but not by DCAs was introduced in [10] and we also do not know whether AfCAs can solve this promise problem.

4 Las Vegas algorithms

In [11], some promise problems were given in order to show the superiority of two-way QFAs (2QCFAs) over two-way PFAs (2PFAs). We show that the same problem can be solved by realtime Las Vegas AfAs or AfAs with restart in linear expected time.

First we review the results given in [11]. Let $\texttt{PAL} = \{w \in \{1,2\}^* \mid w = w^r\}$ be the language of palindromes. Based on PAL, the following promise problem is defined: PAL-NPAL composed of

- $\texttt{PAL-NPAL}_{\texttt{yes}} = \{x0y \mid x \in \texttt{PAL}, y \notin \texttt{PAL}\}$ and
- $\texttt{PAL-NPAL}_{\texttt{no}} = \{x0y \mid x \notin \texttt{PAL}, y \in \texttt{PAL}\}$.

It was shown that PAL-NPAL can be recognized by 2QCFAs exactly but in exponential time and bounded-error 2PFAs can recognize PAL-NPAL only if they are allowed to use a logarithmic amount of memory. Now we show that PAL-NPAL can be recognized by realtime Las Vegas AfAs and so also by AfAs with restart in linear expected time.

Theorem 4. *The promise problem* PAL-NPAL *can be solved by Las Vegas AfA A with any success probability $p < 1$.*

Proof. It is known that AfAs can recognize PAL with one-sided bounded-error [15, 19] and so we can design a Las Vegas automaton for PAL-NPAL by using similar ideas given in [11, 6].

The automaton A has 5 states $S = \{s_1, \ldots, s_5\}$ where s_1 and s_2 are accepting states; s_3 and s_4 are rejecting states; and s_5 is the only neutral state. After reading ¢, the affine state is set to $v_1 = (0\ 0\ 1\ 0\ 0)^T$. Remember that $e(u)$ denotes the encoding of the string $u \in \{1,2\}^*$ in base-3.

We apply the following operators when reading symbols 1 and 2:

$$M_1 = \begin{pmatrix} 4 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -3 & -1 & -4 & -1 & 0 \end{pmatrix} \text{ and } M_2 = \begin{pmatrix} 5 & 2 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -4 & -2 & -6 & -2 & -1 \end{pmatrix}$$

that encode strings u and u^r into the values of the first and second states in base-3 after reading $u \in \{1,2\}^*$. Here the third entry helps for encoding u^r, the fourth entry is irrelevant to encoding, and the fifth entry is used to make the state a well-defined affine vector. By using induction, we can show that M_1 and M_2 do the aforementioned encoding if the first three entries are respectively 0, 0, and 1.

For $u = 1$ or $u = 2$, we can have respectively

$$v_{|\cent 1|} = \begin{pmatrix} 1 \\ 1 \\ 3 \\ * \\ \overline{1} \end{pmatrix} \text{ and } v_{|\cent 2|} = \begin{pmatrix} 2 \\ 2 \\ 3 \\ * \\ \overline{1} \end{pmatrix}.$$

Suppose that u is read, then we have the following affine state

$$v_{|\cent u|} = \begin{pmatrix} e(u) \\ e(u^r) \\ 3^{|u|} \\ * \\ \overline{1} \end{pmatrix}.$$

By using this, we can calculate the new affine states after reading $u1$ and $u2$ as

$$v_{|\cent u1|} = \begin{pmatrix} 3e(u)+1 = e(u1) \\ e(u^r)+3^{|u|} = e(1u^r) \\ 3^{|u1|} \\ * \\ \overline{1} \end{pmatrix} \text{ and } v_{|\cent u2|} = \begin{pmatrix} 3e(u)+2 = e(u2) \\ e(u^r)+2\cdot 3^{|u|} = e(2u^r) \\ 3^{|u2|} \\ * \\ \overline{1} \end{pmatrix},$$

respectively. Thus, our encoding works fine.

Let $x0y$ be the input as promised. Then, before reading the symbol 0, the affine state will be

$$v_{|\cent x|} = \begin{pmatrix} e(x) \\ e(x^r) \\ 3^{|x|} \\ 0 \\ \overline{1} \end{pmatrix}.$$

For the symbol 0, we apply the following operator:

$$M_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

After reading 0, the new affine state will be

$$v_{|\cent x0|} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ e(x)-e(x^r) \\ \overline{1} \end{pmatrix},$$

where the first three entries are set to 0, 0, and 1 for encoding y, and, the difference $e(x)-e(x^r)$ is stored into the fourth entry.

Similarly to above, after reading y, the affine state will be

$$v_{|\cent x0y|} = \begin{pmatrix} e(y) \\ e(y^r) \\ 3^{|y|} \\ e(x)-e(x^r) \\ \overline{1} \end{pmatrix}.$$

Then, the end-marker is read before the weighting operator is applied. Let k be an integer parameter. The affine operator for the symbol \$ is

$$M_{\$}(k) = \begin{pmatrix} k & -k & 0 & 0 & 0 \\ -k & k & 0 & 0 & 0 \\ 0 & 0 & 0 & k & 0 \\ 0 & 0 & 0 & -k & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

and so the final state will be

$$v_f = \begin{pmatrix} k(e(y)-e(y^r)) \\ -k(e(y)-e(y^r)) \\ k(e(x)-e(x^r)) \\ -k(e(x)-e(x^r)) \\ 1 \end{pmatrix}.$$

If the input $x0y$ is a yes-instance, then $x \in \mathtt{PAL}$ and $y \in \mathtt{NPAL}$. Thus, $e(x)-e(x^r)$ is zero and $|e(y)-e(y^r)|$ is at least 1. In such a case, after the weighting operator, the input is accepted with probability at least $\frac{2k}{2k+1}$ and the answer of "don't know" is given with probability at most $\frac{1}{2k+1}$.

If the input $x0y$ is a no-instance, then $x \in \mathtt{NPAL}$ and $y \in \mathtt{PAL}$. Thus, $|e(x)-e(x^r)|$ is at least 1 and $e(y)-e(y^r)$ is zero. In such a case, after the weighting operator, the input is rejected with probability at least $\frac{2k}{2k+1}$ and the answer of "don't know" is given with probability at most $\frac{1}{2k+1}$. By picking a sufficiency big k, the success probability can be arbitrarily close to 1. □

Corollary 1. *The promise problem* PAL-NPAL *can be solved by an exact AfA with restart in linear expected time.*

Proof. In the above proof, we change the neutral states to restarting states, and then obtain the desired machine. For any promised input, the input is either only accepted or only rejected. Since the success probability is constant (p), the expected runtime is $\frac{1}{p}|w|$ for the promised input w. □

We conjecture that bounded-error 2QCFAs cannot solve PAL-NPAL and PAL in polynomial time. Moreover, we leave open whether there exists a promise problem (or a language) solvable by bounded-error AfAs but not by 2QCFAs.

5 Bounded-error algorithms

Similar to PAL, the language $\texttt{TWIN} = \{w0w \mid w \in \{1,2\}^*\}$ can be also recognized by one-sided bounded-error AfAs (see also [15]). After making some straightforward modifications, we can show that the language

$$\texttt{TWIN}(t) = \{w_1 0 w_2 0 \cdots 0 w_t 3 w_t 0 \cdots 0 w_2 0 w_1 \mid w_i \in \{1,2\}^*, 1 \leq i \leq t\}$$

for some $t > 0$ can also be recognized by negative one-sided bounded-error AfAs.

Since it is a non-regular language, it cannot be recognized by bounded-error PFAs and QFAs [1]. On the other hand, we can easily give a bounded-error 2QCFA algorithm for $\texttt{TWIN}(t)$ [19] but similarly to PAL it runs in exponential expected time. By using the impossibility proof given for PAL [5, 11], we can also show that $\texttt{TWIN}(t)$ can be recognized by 2PFAs only if augmented with a logarithmic amount of memory. From the literature [12],[3] we also know that this language can be recognized by a DFA having at least k heads, where

$$t \leq \left(\begin{array}{c} k \\ 2 \end{array} \right) \text{ or } k = \left\lceil \sqrt{2t + \frac{1}{4}} - \frac{1}{2} \right\rceil .$$

Moreover, using nondeterminism and additional pushdown store does not help to save a single head [3]. Bounded-error PFAs can recognize $\texttt{TWIN}(t)$ by using two heads but the error increases when t gets bigger [16, 17]. For a fixed error, we do not know any PFA algorithm using a fixed number of heads. The same result is also followed for bounded-error QFAs with a stack. (It is open whether bounded-error PFAs can recognize $\texttt{TWIN}(t)$ by using a stack [18].)

Based on $\texttt{TWIN}(t)$, we define a seemingly harder language MANYTWINS that is defined by the union of all $\texttt{TWIN}(t)$s:

$$\texttt{MANYTWINS} = \bigcup_{t=1}^{\infty} \texttt{TWIN}(t).$$

Since the number t is not known in advance, we do not know how to design a similar algorithm for the affine, quantum, and classical models discussed above. On the other hand, this language seems a good representative example for how a counter helps for AfAs.

Theorem 5. *The language* MANYTWINS *can be recognized by an AfCA A with one-sided bounded-error arbitrarily close to zero.*

Proof. The automaton A has 10 states: $\{s_1, s_2, s_3, s_1', s_2', s_3', s_e, s_e', s_a, s_r\}$, s_1 is the initial state, and s_a is the only accepting state.

Let k be an arbitrarily big integer. If there is no symbol 3, then the automaton A never switches to the state s_a and so the input is accepted with zero probability. We assume then the input has at least one symbol 3 from now on.

The automaton A stays in s_1 without changing the value of the counter when reading ¢. Then, until reading the first 3, it uses the following transitions.

Let $u_1 = w_1 0 w_2 0 \cdots 0 w_t 3$ be the prefix of the input until the first 3, where $w_i \in \{1,2\}^*$ for each $i \in \{1, \ldots, t\}$ and $t \geq 1$.

When reading a block of $\{1,2\}^*$, say w_i, before a symbol 0 or the symbol 3, it encodes w_i into the value of s_2 in base-3 by help of the states s_1 and s_3. If w_i is the empty string, then the value of s_2 becomes

[3]In the original language, there is a symbol 0 instead of the symbol 3. But since t is fixed, the middle 0 can be easily detected by using internal states and so the results regarding the original language still hold for this modified version.

0. During encoding, the value of s_1, which is 1, does not change and the value of s_3 is updated to have a well-formed affine state.

After reading a 0:

- It stays in s_1 and increases the value of the counter by 1.
- The value of s_2 is $e(w_i)$ before the transition. Then the values of s_e and s'_e are set to $ke(w_i)$ and $-ke(w_i)$, respectively, and the value of the counter does not change. Moreover, the value of s_2 is set to zero.
- Due to the above transitions, the value of s_3 is automatically set to zero.

After reading the first 3:

- It switches from s_1 to s'_1 without changing the value of the counter.
- The value of s_2 is $e(w_t)$ before the transition. Then the values of s_e and s'_e are set to $ke(w_t)$ and $-ke(w_t)$, respectively, and the value of the counter does not change. Moreover, the value of s_2 is set to zero.
- Due to the above transitions, the value of s_3 is automatically set to zero.

Then, after reading u_1, the affine state will be

$$\langle s'_1, t-1\rangle + \sum_{i=1}^{t} \big(ke(w_i)\langle s_e, i-1\rangle - ke(w_i)\langle s'_e, i-1\rangle\big),$$

where, by using the different values of the counter, k times the encoding of each w_i is stored as the values of s_e and s'_e. If $t=0$ ($u_1 = 3$), then the affine state will be $\langle s'_1, 0\rangle$.

If after reading u_1 the automaton reads another symbol 3, then it switches to s_r from s'_1, s'_2, and s'_3, and then stays there until the end of the computation. Thus, in such a case, the input is also accepted with zero probability. Therefore, in the last part, we assume that the input does not have another symbol 3.

Let $u_2 = w'_z 0 w'_{z-1} 0 \cdots 0 w'_1 \$$ be the part to be read after the symbol 3, where $w'_j \in \{1,2\}^*$ for each $j \in \{1,\ldots,z\}$ and $z > 0$. With a similar strategy, when reading a block of $\{1,2\}^*$, say w'_j, before a symbol 0 or the symbol \$, the automaton encodes it into the value of s'_2 in base 3 by the help of states s'_1 and s'_3. If w'_j is the empty string, then the value of s'_2 is zero. During the encoding, the value of s'_1, which is 1, does not change and the value of s'_3 is updated to have a well-formed affine state.

After reading a 0:

- It stays in s'_1 and decreases the value of the counter by 1.
- The value of s'_2 is $e(w'_j)$ before the transition. Then the values of$-ke(w'_j)$ and $ke(w'_j)$ are added to s_e and s'_e, respectively, and the value of the counter does not change. Moreover, the value of s'_2 is set to zero.
- Due to the above transitions, the value of s'_3 is automatically set to zero.

After reading the \$:

- It switches from s'_1 to s_a without changing the value of the counter.
- The value of s'_2 is $e(w'_1)$ before the transition. Then the values of $-ke(w'_1)$ and $ke(w'_1)$ are added to s_e and s'_e, respectively, and the value of the counter does not change. Moreover, the value of s'_2 is set to zero.

- Due to the above transitions, the value of s'_3 is automatically set to zero.

If $z = 0$, then the only transition is switching from s'_1 to s_a.

Suppose that $t = z > 0$. Then it is clear that if $w_t = w'_z$, then the values of the affine state $\langle s_e, t-1\rangle$ and $\langle s'_e, t-1\rangle$ will be set to zero. Otherwise, their values will respectively be $k(e(w_t) - e(w'_z))$ and $-k(e(w_t) - e(w'_z))$, the absolute value of each will be at least k. The same situation holds for each pair (w_i, w'_j) where $1 \leq i = j \leq t$. That means, if the input is a member (including the case of $t = z = 0$), then the final affine state will be $\langle s_a, 0\rangle$ and so the input is accepted with probability 1.

On the other hand, if the input is not a member, then the final affine state will have some non-zero coefficients as the values of some configuration like $\langle s_e, l\rangle$ and $\langle s'_e, l\rangle$ for some l. As described above, the absolute values of these non-zero coefficients are at least k. Thus, any non-member will be accepted with probability at most $\frac{1}{2k+1}$. By picking a sufficiency big k, the success probability can be arbitrarily close to 1. □

In the algorithm given in the proof, the status of the counter is never checked and for each member the value of the counter is set to zero. Thus, it is indeed a blind counter algorithm ([7]): The status of the counter is never checked during the computation and the input is accepted only if the value of the counter is zero at the end of computation. If the value of the counter is non-zero, the input is automatically rejected regardless of the state.

6 Concluding remarks

We introduced affine counter automata as an extended model of affine finite automata, and showed a separation result between exact affine and deterministic models. We also showed that a certain promise problem, which cannot be solved by bounded-error 2PFAs with sublogarithmic space and is also conjectured not to be solved by two-way quantum finite automata in polynomial time, can be solved by Las Vegas affine finite automata in linear time. Lastly, we showed that a counter helps for AfAs by showing that `MANYTWINS`, which is conjectured not to be recognized by affine, quantum or classical finite state models in polynomial time, can be recognized by affine counter automata with one-sided bounded-error in realtime read mode. Since AfCAs are quantum like computation models that can use negative values, we believe that AfCAs can well characterize quantum counter automata and it remains as a future work.

Acknowledgements

Nakanishi was supported by JSPS KAKENHI Grant Numbers 24500003, 24106009 and 16K00007, and also by the Asahi Glass Foundation. Khadiev, Vihrovs, and Yakaryılmaz were supported by ERC Advanced Grant MQC. Prusis was supported by the Latvian State Research Programme NeXIT project No. 1.

References

[1] Andris Ambainis & Abuzer Yakaryılmaz (To appear): *Automata: From Mathematics to Applications*, chapter Automata and Quantum Computing. Available at `http://arxiv.org/abs/1507.01988`.

[2] Aleksandrs Belovs, Juan Andres Montoya & Abuzer Yakaryılmaz (2016): *Can one quantum bit separate any pair of words with zero-error?* Technical Report. Available at `http://arxiv.org/abs/1602.07967`.

[3] Marek Chrobak & Ming Li (1988): *$k+1$ Heads are Better than k for PDAs*. Journal of Computer and System Sciences 37, pp. 144–155, doi:10.1016/0022-0000(88)90004-9.

[4] Alejandro Díaz-Caro & Abuzer Yakaryılmaz (2016): *Affine Computation and Affine Automaton*. In: Computer Science - Theory and Applications, Lecture Notes in Computer Science 9691, Springer, pp. 146–160, doi:10.1007/978-3-319-46976-8. Also available as `http://arxiv.org/abs/1602.04732`.

[5] Cynthia Dwork & Larry Stockmeyer (1992): *Finite state verifiers I: The power of interaction*. Journal of the ACM 39(4), pp. 800–828, doi:10.1145/146585.146599.

[6] Viliam Geffert & Abuzer Yakaryılmaz (2015): *Classical Automata on Promise Problems*. Discrete Mathematics & Theoretical Computer Science 17(2), pp. 157–180, doi:10.1007/978-3-319-09704-6_12.

[7] S. A. Greibach (1978): *Remarks on Blind and Partially Blind One-Way Multicounter Machines*. Theoretical Computer Science 7, pp. 311–324, doi:10.1016/0304-3975(78)90020-8.

[8] Mika Hirvensalo, Etienne Moutot & Abuzer Yakaryılmaz (2017): *On the Computational Power of Affine Automata*. In: Language and Automata Theory and Applications, Lecture Notes in Computer Science 10168, pp. 405–417, doi:10.1007/978-3-319-41312-9_10.

[9] Rishat Ibrahimov, Kamil Khadiev, Krisjanis Prusis, Jevgenijs Vihrovs & Abuzer Yakaryılmaz (2017): *Zero-Error Affine, Unitary, and Probabilistic OBDDs*. Technical Report. Available at `http://arxiv.org/abs/1703.07184`.

[10] Masaki Nakanishi & Abuzer Yakaryılmaz (2015): *Classical and Quantum Counter Automata on Promise Problems*. In: Implementation and Application of Automata, LNCS 9223, Springer, pp. 224–237, doi:10.1007/978-3-319-22360-5_19.

[11] Jibran Rashid & Abuzer Yakaryılmaz (2014): *Implications of quantum automata for contextuality*. In: Implementation and Application of Automata, LNCS 8587, Springer, pp. 318–331, doi:10.1007/978-3-319-08846-4_24.

[12] Arnold L. Rosenberg (1966): *On multi-head finite automata*. IBM Journal of Research and Development 10(5), pp. 388–394, doi:10.1147/rd.105.0388.

[13] A. C. Cem Say & Abuzer Yakaryılmaz (2014): *Quantum Finite Automata: A Modern Introduction*. In: Computing with New Resources, LNCS 8808, Springer International Publishing, pp. 208–222, doi:10.1007/978-3-319-13350-8_16.

[14] A. C. Cem Say & Abuzer Yakarylmaz (2012): *Quantum counter automata*. International Journal of Foundations of Computer Science 23(5), pp. 1099–1116, doi:10.1016/S0304-3975(01)00412-1.

[15] Marcos Villagra & Abuzer Yakaryılmaz (2016): *Language Recognition Power and Succinctness of Affine Automata*. In: Unconventional Computation and Natural Computation, Lecture Notes in Computer Science 9726, Springer, pp. 116–129, doi:10.1007/978-3-319-34171-2_11.

[16] Abuzer Yakaryilmaz (2011): *Superiority of One-Way and Realtime Quantum Machines and New Directions*. In: Third Workshop on Non-Classical Models for Automata and Applications - NCMA 2011, books@ocg.at 282, Austrian Computer Society, pp. 209–224. Available at `http://arxiv.org/abs/1102.3093v1`.

[17] Abuzer Yakaryılmaz (2012): *Superiority of one-way and realtime quantum machines*. RAIRO - Theoretical Informatics and Applications 46(4), pp. 615–641, doi:10.1051/ita/2012018.

[18] Abuzer Yakaryılmaz, Rūsiņš Freivalds, A. C. Cem Say & Ruben Agadzanyan (2012): *Quantum computation with write-only memory*. Natural Computing 11(1), pp. 81–94, doi:10.1007/s11047-011-9270-0.

[19] Abuzer Yakaryılmaz & A. C. Cem Say (2010): *Succinctness of two-way probabilistic and quantum finite automata*. Discrete Mathematics and Theoretical Computer Science 12(2), pp. 19–40. Also available as `http://arxiv.org/abs/0903.0050`.

[20] Abuzer Yakaryılmaz & A. C. Cem Say (2013): *Proving the Power of Postselection*. Fundamenta Informaticae 123(1), pp. 107–134. Also availbale as `http://arxiv.org/abs/1111.3125`.

[21] Sheng Yu (1989): *A pumping lemma for deterministic context-free languages*. Information Processing Letters 31(1), pp. 47–51, doi:10.1016/0020-0190(89)90108-7.