Theory of One Tape Linear Time Turing Machines *

Kohtaro Tadaki[1]† Tomoyuki Yamakami[2] Jack C. H. Lin[2]

[1] ERATO Quantum Computation and Information Project
Japan Science and Technology Corporation, Tokyo, 113-0033 Japan
[2] School of Information Technology and Engineering
University of Ottawa, Ottawa, Ontario, Canada K1N 6N5

Abstract. A theory of one-tape linear-time Turing machines is quite different from its polynomial-time counterpart since one-tape linear-time Turing machines are closely related to finite state automata. This paper discusses structural-complexity issues of one-tape Turing machines of various types (deterministic, nondeterministic, reversible, alternating, probabilistic, counting, and quantum Turing machines) that halt in linear time, where the running time of a machine is defined as the height of its computation tree. We clarify how the machine types affect the computational patterns of one-tape linear-time Turing machines.

Key words. one-tape Turing machine, crossing sequence, finite state automaton, regular language, one-way function, low set, many-one reducibility

1 Prologue

Computer science has revolved around the study of computation incorporated with the analysis and development of algorithms. The notion of a Turing machine, proposed by Turing [33, 34] and independently by Post [27] in the mid 1930s, is now regarded as a mathematical model of an existing computer. This Turing machine model has been a basis of extensive studies in computational complexity theory. Earlier research unearthed the significance of various restrictions on *resources* that are available to a Turing machine: for instance, the number of work tapes, the number of heads, execution time bounds, memory space bounds, and machine types. From a practical aspect, this paper is focused on the liner-time computations and aims at the understanding of how various restrictions directly affect the pattern of computations.

Even for a linear-time bounded Turing machine, the number of work tapes in use and its machine type significantly alter their computational power. For instance, two-tape Turing machines are shown to be more powerful than any one-tape Turing machines [28, 9]. Upon a multiple-tape Turing machine, Paul, Pippenger, Szemeredi, and Trotter [26] proved in early 1980s that linear-time nondeterministic Turing machines are more powerful than their deterministic counterparts.

Of particular interest in this paper is a one-tape (or single-tape) linear-time Turing machine. Not surprisingly, this simple model proves a close tie to finite automata. Despite its simplicity, such a model still offers a complex structure of computation patterns. A theory of one-tape linear-time complexity nonetheless draws a picture quite different from multiple-tape models and polynomial-time models. We can prove the collapses and separations of many one-tape liner-time complexity classes without any unproven assumption, such as the existence of one-way functions.

Hennie [14] made the first major contribution to the theory of one-tape linear-time Turing machines in the mid 1960s. He demonstrated that no one-tape linear-time deterministic Turing machine can be more powerful than a deterministic finite state automaton. To prove his result, Hennie described the behaviors of a Turing machine in terms of the sequential changes of the machine's internal states at the time when the tape head crosses a boundary of two adjacent tape cells. Such a sequence of states is called a *crossing sequence* at a boundary. He argued that (i) any one-tape linear-time deterministic Turing machine has short crossing sequences at every boundary and (ii) if any crossing sequence of the machine is short, then this machine recognizes only a regular language. In terms of the non-regularity measure of Dwork and Stockmeyer [10], the second claim asserts that any language accepted by a machine with short crossing sequences has constantly-bounded non-regularity. Later, Kobayashi [20] extended Hennie's argument to show that any language recognized by a one-tape $o(n \log n)$-time

*An extended abstract appeared in the Proceedings of the 30th SOFSEM Conference on Current Trends in Theory and Practice of Computer Science, Lecture Notes in Computer Science, Vol.2932, pp.335–348, Springer-Verlag, January 24–30, 2004. This work was in part supported by the Natural Sciences and Engineering Council of Canada.

†Present address: 21st Century COE Program: Research on Security and Reliability in Electronic Society, Chuo University, 1-13-27 Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan. This work was partly done while he was visiting the University of Ottawa between November 1 and December 1, 2001.

deterministic Turing machine is also regular. This time bound $o(n \log n)$ is optimal since certain one-tape $O(n \log n)$-time deterministic Turing machines can recognize non-regular languages.

Unlike polynomial-time computation, one-tape linear-time nondeterministic computation is sensitive to the definition of the machine's running time. Such sensitivity is also observed in average-case complexity theory [36]. By taking the *weak definition* that defines the running time of a nondeterministic Turing machine to be the length of a "shortest" accepting path, Michel [25] demonstrated that one-tape linear-time nondeterministic Turing machines solve even NP-complete problems. Therefore, the weak definition gives an enormous power to one-tape nondeterministic machines. On the contrary, the *strong definition* (in Michel's term) requires the running time to be the length of any "longest" computation path. This strong definition provides us with a reasonable basis to study the effect of linear-time bounded computations. We therefore adopt the strong definition in this paper. By expanding Kobayashi's result, we prove that one-tape $o(n \log n)$-time nondeterministic Turing machines recognize only regular languages.

Alternating Turing machines of Chandra, Kozen, and Stockmeyer [4] naturally expand nondeterministic machines. The number of alternations of an alternating Turing machine seems to alter the computational power of the machine. In particular, we demonstrate that a constant number of alternations do not provide any additional computational power to a one-tape linear-time alternating Turing machine; that is, such a machine cannot recognize any non-regular language.

Probabilistic Turing machines with fair coin tosses of Gill [12], however, present distinctive features. A language recognized by a certain one-head one-way probabilistic finite automaton (with unbounded-error probability) is known as a *stochastic* language [28]. By employing a crossing sequence argument, we show that any language recognized by a one-tape linear-time probabilistic Turing machine with unbounded-error probability is just stochastic. Our collapse result proves a close relationship between one-tape linear-time Turing machines and finite state automata.

The Turing machine model, nonetheless, presents distinctive looks when we discuss functions rather than languages. Beyond the framework of formal language theory, Turing machines are capable of computing (partial multi-valued) functions by simply modifying their tape contents. Such functions also serve as many-one reductions between two languages. In this paper, we introduce various types of "many-one one-tape linear-time" reductions. In particular, nondeterministic many-one reducibility plays an important role in showing the collapse of alternating linear-time complexity classes. Naturally, we can view many-one reducibility as oracle mechanism of the simplest form. In terms of such oracle computation, we can easily prove the existence of an oracle that separates the one-tape linear-time nondeterministic complexity class from its deterministic counterpart.

The existence of a one-way function is a key to the building of secure cryptosystems. Restricted to length-preserving functions, we show that no length-preserving one-way function is computed by one-tape deterministic Turing machines in linear time, where "length-preserving" means that the output size of a function is the same as its input size. Therefore, there is no one-way permutation computable in linear time by any one-tape deterministic Turing machine.

The number of accepting computation paths of a time-bounded nondeterministic Turing machine has played a significant role in computational complexity theory. Valiant [35] introduced the notion of a counting Turing machine to count such a number. Counting Turing machines have been used to study the complexity of "counting." The functions computed by these machines are called *counting functions* and complexity classes of languages defined in terms of counting functions are generally called *counting classes*. We show that counting functions computable by one-tape linear-time counting Turing machines are more powerful than deterministically computable functions. By contrast, we also prove that certain counting classes induced from one-tape linear-time counting Turing machines collapse to the collection of regular languages.

The latest variant of the Turing machine model is a quantum Turing machine, which is seen as an extension of a probabilistic Turing machine. While a probabilistic Turing machine is based on classical physics, a quantum Turing machine is based on quantum physics. The notion of such machinery was introduced by Deutsch [7] and reformulated by Bernstein and Vazirani [3]. Of all the known types of quantum Turing machines, we study only the following two machine types: bounded-error quantum Turing machines [3] and "nondeterministic" quantum Turing machines [1]. We give a characterization of one-tape linear-time "nondeterministic" quantum Turing machines in terms of counting Turing machines.

We also discuss a supplemental mechanism called *advice* to enhance the computational power of a Turing machines. A piece of advice is meant for additional information supplied to underlying computation besides an input [18]. We adapt the notion of advice into one-tape Turing machines. We demonstrate a context-free language that cannot be recognized by any one-tape linear-time deterministic Turing machine with advice.

2 Model of Computation: Turing Machines

This paper uses a standard definition of a Turing machine (see, e.g., [16, 8]) as a computational model. Of special interest is a one-tape Turing machine. Here, we give fundamental notions and notation associated with our computational model.

Let $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ be the sets of all integers, of all rational numbers, of all real numbers, respectively. In particular, let $\mathbb{R}^{\geq 0}$ be $\{r \in \mathbb{R} \mid r \geq 0\}$. Moreover, let $\mathbb{N}$ denote the set of all natural numbers (i.e., non-negative integers) and set $\mathbb{N}^{+} = \mathbb{N} - \{0\}$. For any two integers n, m with $n \leq m$, $[n, m]_{\mathbb{Z}}$ denotes the set $\{n, n+1, n+2, \ldots, m\}$. We assume that all logarithms are to the base two. Throughout this paper, we use the notation Σ (Σ_1, Σ_2, etc.) to denote an arbitrary nonempty finite alphabet. A *string* over alphabet Σ is a finite sequence of elements from Σ. In particular, the *empty string* over any alphabet is always denoted λ. Let Σ^* denote the collection of all finite strings over Σ. For a string x in Σ^*, $|x|$ denotes the *length* of x (i.e., the number of symbols in x). A *language* (or simply a "set") *over alphabet* Σ is a subset of Σ^*, and a *complexity class* is a collection of certain languages. The *complement* of A is $\Sigma^* - A$ and often denoted $\overline{A}$ if Σ is clear from the context. For any complexity class $\mathcal{C}$, the notation co-$\mathcal{C}$ denotes the *complement* of $\mathcal{C}$; that is, the collection of all languages whose complements belong to $\mathcal{C}$.

We often use *multi-valued partial functions* as well as single-valued total functions. For any multi-valued partial function f mapping from a set D to a set E, $\mathrm{dom}(f)$ denotes the *domain* of f, namely, $\mathrm{dom}(f) = \{x \in D \mid f(x) \text{ is defined}\}$ and, for each $x \in \mathrm{dom}(f)$, $f(x)$ is a subset of E. If f is specially single-valued, we write "$f(x) = y$" instead of "$y \in f(x)$" by identifying the set $\{y\}$ with y itself. A multi-valued partial function f from Σ_1^* to Σ_2^* is called *length-preserving* if, for every $x \in \Sigma_1^*$ and $y \in \Sigma_2^*$, $y \in f(x)$ implies $|y| = |x|$. For convenience, we write LPF to denote the collection of all length-preserving multi-valued partial functions from Σ_1^* to Σ_2^*, where Σ_1 and Σ_2 are arbitrary nonempty finite alphabets. Note that any *total* function is also a partial function. The *characteristic function* χ_A of a language A over Σ is defined as, for any string x in Σ^*, $\chi_A(x) = 1$ if $x \in A$ and $\chi_A(x) = 0$ otherwise. For any single-valued total function g from $\mathbb{N}$ to $\mathbb{N}$, $O(g(n))$ denotes the set of all single-valued total functions f such that $f(n) \leq c \cdot g(n)$ for all but finitely many numbers n in $\mathbb{N}$, where c is a positive constant independent of n. Similarly, $o(g(n))$ is the set of all functions f such that, for every positive constant c, $f(n) < c \cdot g(n)$ for all but finitely many numbers n in $\mathbb{N}$.

We give the basic definition of one-tape Turing machines. A *one-tape Turing machine* (abbreviated 1TM) is a septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$, where Q is a finite set of (internal) states, Σ is a nonempty finite input alphabet, Γ is a finite tape alphabet including Σ, q_0 in Q is an initial state, q_{acc} and q_{rej} in Q are an accepting state and a rejecting state, respectively, and δ is a transition function. In later sections, we give different types of transition functions δ, which give rise to various types of 1TMs. Notice that certain types of 1TMs require additional sets and functions. A *halting state* is either q_{acc} or q_{rej}. Our 1TM is equipped only with one input/work tape such that (i) the tape stretches infinitely to both ends, (ii) the tape is sectioned by cells, and (iii) all cells in the tape are indexed with integers. The tape head starts at the cell indexed 0 (called the *start cell*) and either moves to the right (R), moves to the left (L), or stays still (N).

A *configuration* of a 1TM M, which represents a snapshot of a "computation," is a triplet of an internal state, a head position, and a tape content of M. The initial configuration of M on input x is the configuration in which M is in internal state q_0 with the head scanning the start cell and the string x is written in an input/work tape, surrounded by the blank symbols, in such a way that the leftmost symbol of x is in the start cell. A computation of a 1TM M generally forms a tree (called a *computation tree*) whose nodes are certain configurations of M. The root of such a computation tree is an initial configuration, leaves are final configurations, and every non-root node is obtained from its parent node by a single application of δ. Each path of a computation tree, from its root to a certain leaf is referred to as a *computation path*. An *accepting* (a *rejecting*, a *halting*, resp.) *computation path* is a path terminating in an accepting (a rejecting, a halting, resp.) configuration. We say that a TM *halts* on input x if *every* computation path of M on input x eventually reaches a certain halting state. Of particular importance is the synchronous notion for a 1TM. A 1TM is said to be *synchronous* if all computation paths terminate at the same time on each input. In addition, a 1TM is *dynamic* if its head always moves without staying still.

Throughout this paper, we use the term "*running time*" for a 1TM M taking input x, denoted $\mathrm{Time}_M(x)$, to mean the height of the computation tree produced by the execution of M on input x; in other words, the length of the longest computation path of M on x. We often use $T(n)$ to denote a time-bounding function of a given 1TM that maps from $\mathbb{N}$ to $\mathbb{N}$. Furthermore, a "linear function" means a function of the form $cx + d$ for a certain constant $c, d \in \mathbb{R}^{\geq 0}$. A 1TM M is said to run in *linear time* if its running time $\mathrm{Time}_M(x)$ on any input x is bounded above by $f(|x|)$ for a certain linear function f.

Although our machine has only one input/work tape, the tape can be split into a constant number of *tracks*.

To describe such tracks, we use the following notation. For any pair of symbols $a, b \in \Sigma$, $[\begin{smallmatrix} a \\ b \end{smallmatrix}]$ denotes the special tape symbol for which a is written in the upper track and b is written in the lower track of the same cell. By extending this notion, for any strings $x, y \in \Sigma^*$ with $|x| = |y|$, we write $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$ to denote the concatenation $[\begin{smallmatrix} x_1 \\ y_1 \end{smallmatrix}][\begin{smallmatrix} x_2 \\ y_2 \end{smallmatrix}]\cdots[\begin{smallmatrix} x_n \\ y_n \end{smallmatrix}]$ if $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_n$, where all x_i's and y_i's are in Σ.

For any types of 1TM, certain *accepting criteria* and *rejecting criteria* are usually imposed to define the set of "accepted" input strings. In general, a language A is said to be *recognized* by a 1TM M of certain machine type if, for every string x, (i) if $x \in A$ then M halts on input x and satisfies the given accepting criteria and (ii) if $x \notin A$ then M halts and satisfies the rejecting criteria.

The non-regularity measure plays an important role. For any pair x and y of strings and any integer $n \in \mathbb{N}$, we say that x and y are *n-dissimilar* with respect to a given language L if there exists a string z such that (i) $|xz| \leq n$ and $|yz| \leq n$ and (ii) $xz \in L \iff yz \notin L$. For each $n \in \mathbb{N}$, define $N_L(n)$ (the *non-regularity* measure of L at n) to be the maximal cardinality of the set in which any distinct pair is n-dissimilar with respect to L [10]. It is immediate from the Myhill-Nerode theorem [16] that a language L is regular if and only if $N_L(n) = O(1)$ [10]. This is further improved by the results of Karp [17] and of Kaneps and Freivalds [19] as follows: a language L is regular if and only if $N_L(n) \leq \frac{n}{2} + 1$ for all but finitely-many numbers n in $\mathbb{N}$.

We assume the reader's familiarity with the notion of *finite (state) automata* (see, e.g., [15, 16]). The class of all *regular* languages is denoted REG, where a language is called *regular* if it is recognized by a certain (one-head one-way) deterministic finite automaton. The languages recognized by (one-head one-way) push-down automata are called *context-free* and the notation CFL denotes the collection of all context-free languages.

A *rational (one-head) one-way generalized probabilistic finite automaton* (for short, *rational 1GPFA*) [30, 32] is a quintuple $N = (S, \Sigma, \pi, \{T(\sigma) \,|\, \sigma \in \Sigma\}, \eta)$, where (i) S is a finite set of states, (ii) Σ is a finite alphabet, (iii) π is a row vector which has $|S|$ rational components, (iv) for each $\sigma \in \Sigma$, $T(\sigma)$ is an $|S| \times |S|$ matrix whose elements are rational numbers, and (v) η is a column vector of $|S|$ rational entries. A *word matrix* $T(x)$ of N on input string $x \in \Sigma^*$ is defined as $T(\lambda) = I$ for the empty string λ, where I is the identity matrix of order $|S|$ and $T(x_1 \ldots x_k) = T(x_1) \ldots T(x_k)$ for $x_1, \ldots, x_k \in \Sigma$. For each $x \in \Sigma^*$, the *acceptance function* $p_N(x)$ is defined to be $\pi\, T(x)\, \eta$. By $1\mathrm{GAF}_{rat}$, we denote the set of all acceptance functions of rational 1GPFAs. A matrix T is called *stochastic* if every row of T sums up to exactly 1. A *rational (one-head) one-way probabilistic finite automaton* (for short, *rational 1PFA*) [28] N is a rational 1GPFA $(S, \Sigma, \pi, \{T(\sigma) \,|\, \sigma \in \Sigma\}, \eta)$ such that (i) π is a stochastic row vector whose entries are all nonnegative, (ii) for each $\sigma \in \Sigma$, $T(\sigma)$ is stochastic with nonnegative components, and (iii) η is a column vector whose components are either 0 or 1. From this η, we define the set F of all final states of N as $F = \{a \in S \mid \text{the } a\text{th entry of } \eta \text{ is } 1\}$. Moreover, since $p_N(x)$ equals the probability that N accepts x, $p_N(x)$ is particularly called the *acceptance probability* of N on input x.

Let ε be any rational number. For each rational 1GPFA N, let $L(N, \varepsilon) = \{\, x \in \Sigma^* \mid p_N(x) > \varepsilon\}$ and $L^{=}(N, \varepsilon) = \{\, x \in \Sigma^* \mid p_N(x) = \varepsilon\}$, where ε is called a *cut point* of N. Let GSL_{rat} and SL_{rat} be the collections of all sets $L(N, \varepsilon)$ for certain rational 1GPFAs N and for certain rational 1PFAs, respectively, where ε is an arbitrary rational number. Similarly, $\mathrm{GSL}^{=}_{rat}$ and $\mathrm{SL}^{=}_{rat}$ are defined by substituting $L^{=}(N, \varepsilon)$ for $L(N, \varepsilon)$. Sets in SL_{rat} are known as *stochastic* languages [28]. Turakainen [32] demonstrated the equivalence of GSL_{rat} and SL_{rat}. With a similar idea, we can show that $\mathrm{GSL}^{=}_{rat} = \mathrm{SL}^{=}_{rat}$. The proof of this claim is left to the avid reader.

3 Deterministic and Reversible Computations

Of all computations, deterministic computation is the most intuitive type of computation. We begin with reviewing the results of Hennie [14] and Kobayashi [20] on one-tape deterministic Turing machines. A *deterministic 1TM*, embodying a sequential computation, is formally defined by a transition function δ that maps $(Q - \{q_{acc}, q_{rej}\}) \times \Gamma$ to $Q \times \Gamma \times \{L, N, R\}$. Since the notation DLIN is widely used for the model of multiple-tape linear-time Turing machines, we use the following new notations to emphasize our one-tape model. The general notation 1-DTime($T(n)$) denotes the collection of all languages recognized by deterministic 1TMs running in $T(n)$ time. Given a set $\mathcal{T}$ of time-bounding functions, 1-DTime($\mathcal{T}$) stands for the union of 1-DTime($T(n)$)s over all functions T in $\mathcal{T}$. The *one-tape deterministic linear-time* complexity class 1-DLIN is then defined to be 1-DTime($O(n)$).

Earlier, Hennie [14] proved that REG = 1-DLIN by employing a so-called *crossing sequence argument*. Elaborating Hennie's argument, Kobayashi [20] substantially improved Hennie's result by showing REG = 1-DTime($o(n \log n)$). This bound $o(n \log n)$ is optimal because 1-DTime($O(n \log n)$) contains non-regular languages, e.g., $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{a^{2^n} \mid n \in \mathbb{N}\}$. Therefore, we obtain the fundamental collapse result concerning our one-tape Turing machines.

Proposition 3.1 [14, 20] REG $=$ 1-DTime$(o(n\log n)) \neq$ 1-DTime$(O(n\log n))$.

In the early 1970s, Bennett [2] initiated a study of reversible computation. Reversible computations have recently drawn wide attention from physicists as well as computer scientists in connection to quantum computations. We adapt the following definition of a (deterministic) reversible Turing machine given by Bernstein and Vazirani [3]. A *(deterministic) reversible 1TM* is a deterministic 1TM of which each configuration has at most one predecessor configuration. We use the notation 1-revDTime$(T(n))$ to denote the collection of all languages recognized by $T(n)$-time reversible 1TMs and let 1-revDTime$(\mathcal{T})$ be $\bigcup_{T\in\mathcal{T}}$ 1-revDTime$(T(n))$. Finally, let 1-revDLIN $=$ 1-revDTime$(O(n))$. Obviously, 1-revDLIN $\subseteq$ 1-DLIN.

Kondacs and Watrous [21] recently demonstrated that any one-head one-way deterministic finite automaton can be simulated in linear time by a certain one-head two-way deterministic reversible finite automaton. Since any one-head two-way deterministic reversible finite automaton is indeed a reversible 1TM, we obtain that REG $\subseteq$ 1-revDLIN. Proposition 3.1 thus concludes:

Proposition 3.2 REG $=$ 1-revDLIN $=$ 1-revDTime$(o(n\log n))$.

The computational power of a Turing machine can be enhanced by supplemental information given parallel to input. Karp and Lipton [18] introduced the notion of such extra information under the name of *advice*, which is given depending only on the size of input. Damm and Holzer [6] later considered finite automata that take the Karp-Lipton type advice. To make most of the power of advice, we take a slightly different formulation for our linear-time 1TMs. In this paper, for any complexity class $\mathcal{C}$ based on 1TMs, the notation $\mathcal{C}/n$ is used to represent the collection of all languages A such that there exist an alphabet Σ, a set B in $\mathcal{C}$, and a length-preserving function h from $\mathbb{N}$ to Σ^* (called an *advice function*) such that, for every $x\in\Sigma^*$, $x\in A$ if and only if $[\begin{smallmatrix} x \\ h(|x|) \end{smallmatrix}]\in B$. For instance, any language A of density ≤ 1 (i.e., its density $|A\cap\Sigma^n|$ is always at most 1) clearly belongs to REG$/n$. This gives the obvious separation REG $\subsetneqq$ REG$/n$. However, REG$/n$ is not as large as CFL since, as we see below, the context-free language $L_{num}=\{x\in\{0,1\}^* \mid \#_0(x)=\#_1(x)\}$, where $\#_i(x)$ denotes the number of symbol i in x, is situated outside of REG$/n$. This result will be used in Section 7.

Lemma 3.3 *The language L_{num} is not in* REG$/n$.

Proof. Let $\Sigma=\{0,1\}$. Assuming that $L_{num}\in$ REG$/n$, choose a deterministic finite automaton $M=(Q,\Sigma,q_0,F)$ and an advice function h from $\mathbb{N}$ to Σ^* such that, for every $x\in\Sigma^*$, $x\in L_{num}$ if and only if $[\begin{smallmatrix} x \\ h(|x|) \end{smallmatrix}]\in B$. Take the minimal integer n satisfying $n+1>|Q|$. For each $k\in[0,n]_{\mathbb{Z}}$, define y_k to be any string of length n satisfying $\#_0(y_k)=k$. Hereafter, we focus on strings yz's of length $2n$.

There exist two *distinct* indices $k,l\in[0,n]_{\mathbb{Z}}$ such that (i) $y_kz_k, y_lz_l\in L_{num}$ for some $z_k,z_l\in\Sigma^n$ and (ii) M enters the same internal state after reading y_k as well as y_l. Such a pair (k,l) exists since $n+1>|Q|$. It follows from these conditions that M also accepts the input $[\begin{smallmatrix} y_kz_l \\ h(|y_kz_l|) \end{smallmatrix}]$. Thus, $\#_0(y_kz_l)=\#_1(y_kz_l)$, which implies $\#_0(z_l)=n-k$. However, since $\#_0(y_lz_l)=\#_1(y_lz_l)$, we obtain $\#_0(y_l)=k$. This contradicts the definition of y_l. Therefore, $L_{num}\notin$ REG$/n$. □

Up to now, we have seen Turing machines as language recognizers. Beyond the framework of formal language theory, Turing machines are fully capable of computing partial functions. Note that, since a 1TM M has only one input/work tape, we need to designate the same tape as the output tape. As for an outcome of a machine, we use the following convention. When a machine eventually halts, if the output tape consists only of a single block of non-blank symbols, say s, which is surrounded by the blank symbols, with the condition that the leftmost non-blank symbol should be written in the start cell, then s is considered as the *valid* outcome of the machine.

For notational convenience, we introduce the function class 1-FLIN in the following fashion. A *total* function from Σ_1^* to Σ_2^* is in 1-FLIN if there exists a deterministic 1TM M satisfying that, on any input $x\in\Sigma_1^*$, (i) M halts by entering the accepting state in time linear in $|x|$ and (ii) when M halts, M outputs $f(x)$ as a valid outcome. When "partial" functions are concerned, we conventionally regard the "rejecting state" as an invalid outcome. We thus define 1-FLIN(partial) to be the collection of all partial functions f from Σ_1^* to Σ_2^* such that, for every $x\in\Sigma_1^*$, (i) if $x\in\mathrm{dom}(f)$ then M enters an accepting state with outputting $f(x)$ and (ii) if $x\notin\mathrm{dom}(f)$ then M enters a rejecting state (and we ignore the tape content).

Historically, automata theory has also provided the machinery that can compute functions (see, e.g., [16] for a historical account). We herein consider only so-called Mealy machines. A *Mealy machine* $(Q,\Sigma,\Gamma,q_0,\delta,\nu)$ is a deterministic finite automaton $(Q,\Sigma,\Gamma,q_0,\delta)$, ignoring final states, together with a total function ν from Q to Σ such that, on input $x=x_1x_2\cdots x_n$, it outputs $\nu(q_0,x_1)\nu(q_1,x_2)\cdots\nu(q_{n-1},x_n)$, where $(q_0,q_1,\ldots,q_n)$ is the sequence of states in Q satisfying $\delta(q_{i-1}, x_i) = q_i$ for every $i \in [1, n]_{\mathbb{Z}}$. Note that a Mealy machine computes only length-preserving functions. In the following proposition, we can show the existence of a length-preserving function in 1-FLIN that cannot be computed by any Mearly machines.

Proposition 3.4 *There exists a length-preserving function in* 1-FLIN *that cannot be computed by any Mealy machines.*

Proof. Consider the following length-preserving function f defined by $f(x_1x_2 \cdots x_n) = x_nx_1 \cdots x_{n-1}$ for any $x_1, x_2, \ldots, x_n \in \{0, 1\}$. It is not difficult to design a deterministic 1TM that computes f in linear time and thus f is in 1-FLIN. To complete the proof, we need to show that no Mealy machine computes f. Assume otherwise that f is computed by a Mealy machine $M = (Q, \Sigma, \Gamma, q_0, \delta, \nu)$. Let $n \geq 2$ be any integer and let $x = x_1x_2 \cdots x_n$ be any input, where each x_i is a bit. Since M computes f, $\nu(q_0, x_1) = x_n$. Clearly, x_n depends only on x_1. For another input $x' = x_1x_2 \cdots \overline{x}_n$, where $\overline{a} = 1 - a$, we also obtain $\nu(q_0, x_1) = \overline{x}_n$ by the definition of f. This clearly contradicts the previous equality $\nu(q_0, x_1) = x_n$. Therefore, no Mearly machine computes f. □

4 Nondeterministic Computation

Nondeterminism has been widely studied in the literature since many problems arising naturally in computer science have nondeterministic traits. In a nondeterministic computation, a Turing machine has several choices to follow at each step. We expand the collapse result of deterministic 1TMs in Section 3 into nondeterministic 1TMs. We also discuss the multi-valued partial functions computed by one-tape nondeterministic Turing machines and show how to simulate such functions in a certain deterministic manner.

4.1 Nondeterministic Languages

As a language recognizer, a *nondeterministic 1TM* takes a transition function δ that maps $(Q - \{q_{acc}, q_{rej}\}) \times \Gamma$ to $2^{Q \times \Gamma \times \{L,N,R\}}$, where 2^A denotes the power set of A. An execution of a nondeterministic 1TM produces a computation tree. We say that a nondeterministic 1TM M *accepts* an input x exactly when there exists an accepting computation path in the computation tree of M on input x. Similar to the deterministic case, let 1-NTime($T(n)$) denote the collection of all languages recognized by $T(n)$-time nondeterministic 1TMs and let 1-NTime($\mathcal{T}$) be the union of all 1-NTime($T(n)$) for all $T \in \mathcal{T}$. We define the *one-tape nondeterministic linear-time* class 1-NLIN to be 1-NTime($O(n)$).

We first expand Kobayashi's collapse result on 1-DTime($o(n \log n)$) into 1-NTime($o(n \log n)$).

Theorem 4.1 REG = 1-NTime($o(n \log n)$) $\neq$ 1-NTime($O(n \log n)$).

The proof of Theorem 4.1 consists of two technical lemmas: Lemmas 4.2 and 4.3. The first lemma is a core of Kobayashi's argument in [20, Theorem 3.3] and the second lemma is due to Hennie [14, Theorem 2].

For the description of the lemmas, we wish to introduce the key terminology. Let M be any type of 1TM, which is not necessarily nondeterministic. Any boundary that separates two adjacent cells in M's tape is called an *intercell boundary*. The *crossing sequence* at intercell boundary b along computation path s of M is the sequence of inner states of M at the time when the tape head crosses b, first from left to right, and then alternately in both directions. To visualize the head move, we assume that the head scans tape symbol σ at tape cell i in state p. If we apply a transition $(q, \tau, R) \in \delta(p, \sigma)$, then the machine writes symbol τ into cell i, enters state q, and then moves the head to cell $i + 1$. The state in which the machine crosses the boundary between cell i and cell $i + 1$ is q (not p). Similarly, if we apply a transition $(q, \tau, L) \in \delta(p, \sigma)$, then q is the state in which the machine crosses the boundary between cell $i - 1$ and cell i. The *right-boundary* of x is the intercell boundary between the rightmost symbol of x and its right-adjacent blank symbol. Similarly, the *left-boundary* of x is defined as the intercell boundary between the leftmost symbol of x and its left-adjacent symbol. Any intercell boundary between the right-boundary and the left-boundary of x (including both ends) is called a *critical-boundary* of x.

Lemma 4.2 comes from the close observation of Kobayashi's argument that (i) the acceptance criteria of nondeterministic 1TMs are irrelevant and (ii) the argument is applicable to any other models of 1TMs dealt with in this paper. For completeness, the proof of Lemma 4.2 is included in Appendix.

Lemma 4.2 *Assume that $T(n) = o(n \log n)$. For any $T(n)$-time nondeterministic 1TM M, there exists a constant $c \in \mathbb{N}$ such that, for each string x, any crossing sequence at every intercell boundary in every computation* *path of M on input x has length at most c. In addition, no acceptance criteria of the machine are necessary.*

In essence, Hennie [14] proved that any deterministic computation with short crossing sequences has constantly-bounded non-regularity. We generalize his result to the nondeterministic case as in the following lemma. Different from the previous lemma, Lemma 4.3 relies on the acceptance criteria of nondeterministic 1TMs. Nonetheless, Lemma 4.3 does not refer to rejecting computation paths. For readability, the proof of Lemma 4.3 is also placed in Appendix.

Lemma 4.3 *Let L be any language and let M be any nondeterministic 1TM that recognizes L. For each $n \in \mathbb{N}$, let S_n be the set of all crossing sequences at any critical-boundary along any accepting computation path of M on any input of length $\leq n$. Then, $N_L(n) \leq 2^{|S_n|}$ for all $n \in \mathbb{N}$, where $|S_n|$ denotes the cardinality of S_n.*

Since REG is closed under complementation, so is 1-NTime($o(n \log n)$). In contrast, a simple crossing sequence argument proves that 1-NTime($O(n \log n)$) does not contain the set of palindromes $L_{pal} = \{x \in \{0,1\}^* \mid x = x^R\}$, where x^R is the *reverse* of x. Since $\overline{L_{pal}} \in$ 1-NTime($O(n \log n)$), 1-NTime($O(n \log n)$) is different from co-1-NTime($O(n \log n)$).

Corollary 4.4 *The class* 1-NTime($o(n \log n)$) *is closed under complementation whereas* 1-NTime($O(n \log n)$) *is not closed under complementation.*

Reducibility has played a central role in the theory of NP-completeness and can be seen as a basis of relativization. Turing reducibility, for instance, induces a typical adaptive oracle computation whereas truth-table reducibility represents a nonadaptive (or parallel) oracle computation. Similarly, we introduce the following restricted reducibility into one-tape Turing machines. A set A over alphabet Σ_1 is *many-one 1-NLIN-reducible to* another set B over alphabet Σ_2 (notationally, $A \leq_m^{\text{1-NLIN}} B$) if there exist a linear function $T \in \mathcal{T}$ and a nondeterministic 1TM M such that, for every string x in Σ_1^*, (i) M on input x halts within time $T(|x|)$ with the tape consisting only of one block of non-blank symbols, say y_p, on each computation path p, provided that the first symbol of y_p must be in the start cell, (ii) when M halts, the tape head returns to the start cell along each computation path, and (iii) $x \in A$ if and only if $y_p \in B$ for a certain computation path p of M on input x. We use the notation 1-NLIN$_m^B$ to denote the collection of all languages A that are many-one 1-NLIN-reducible to B. Furthermore, 1-NLIN$_m^{\mathcal{C}}$ stands for the union of 1-NLIN$_m^B$'s over all sets B in $\mathcal{C}$ for any complexity class $\mathcal{C}$.

A straightforward simulation shows that 1-NLIN$_m^{\text{REG}}$ is included in 1-NLIN. More generally, we can show the following proposition. This result will be used in Section 5.

Proposition 4.5 *For any language C,* 1-NLIN$_m^{\text{1-NLIN}_m^C} \subseteq$ 1-NLIN$_m^C$.

Proof. Let A, B, and C be any languages and assume that $A \leq_m^{\text{1-NLIN}} B \leq_m^{\text{1-NLIN}} C$. It suffices to show that $A \leq_m^{\text{1-NLIN}} C$. Take a nondeterministic 1TM M that many-one 1-NLIN-reduces A to B and another nondeterministic 1TM M' that many-one 1-NLIN-reduces B to C. Now, consider the following 1TM N: on input x, simulate M on x and when it halts start M'. This machine N is clearly nondeterministic and its running time is $O(n)$ since so are the running time of M and M'. Using the many-one reduction property, we can show that N reduces A to C. □

Similar to the many-one 1-NLIN-reducibility, we can define the "many-one 1-DLIN-reducibility" and its corresponding relativized class 1-DLIN$_m^B$. The oracle separation is possible between 1-DLIN$_m^B$ and 1-NLIN$_m^B$. The proof can be obtained by a standard diagonalization technique that constructs an oracle separating NP from P. For completeness, we include the proof.

Proposition 4.6 *There exists a recursive oracle B such that* 1-DLIN$_m^B \neq$ 1-NLIN$_m^B$.

Proof. For any $B \subseteq \{0,1\}^*$, define $L_B = \{0^n \mid \exists x \in \{0,1\}^n [x \in B]\}$. Obviously, L_B belongs to 1-NLIN$_m^B$ for any B. It suffices to show that L_B is not in 1-DLIN$_m^B$ for a certain set B. To prove this claim, we first enumerate all deterministic 1TMs as $M_0, M_1, \ldots$ in a certain effective way. Diagonalize all such machines as follows. Let $n_0 = 0$. We construct the desired oracle B by stages. For each stage $s = \langle i, j \rangle \geq 1$, choose n_s to be $\min\{n \in \mathbb{N} \mid n_{s-1} < n \wedge jn + j < 2^n\}$, where $\langle , \rangle$ is the standard paring function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. If $M_i^{B_{s-1}}$ accepts 0^{n_s} in time $jn_s + j$, then let B_s be B_{s-1}. In case where $M_i^{B_{s-1}}$ rejects 0^{n_s}, choose the minimal $x \in \{0,1\}^{n_s}$ such that M_i does not query x to B_{s-1}. Using this x, we define B_s to be $B_{s-1} \cup \{x\}$. Finally, define $B = \bigcup_i B_i$. By the construction of B, B satisfies the proposition. □

4.2 Length-Preserving Multi-Valued Partial Functions

Conventionally, a Turing machine that outputs values is called a *transducer*. Nondeterministic transducers generally compute multi-valued partial functions. Consider any nondeterministic 1TM that outputs a string in Σ_2^* (whose leftmost symbol is in the start cell) on each computation path and enters a halting state. Similar to partial functions introduced in Section 3, we invalidates any rejecting computation path and let $M(x)$ denote the set of all *valid* outcomes of M on input x. In particular, in case where M on input x enters the rejecting state on all computation paths, $M(x)$ is undefined. A multi-valued partial function f from Σ_1^* to Σ_2^* is in 1-NLINMV if there exists a linear-time nondeterministic 1TM M such that $f(x) = M(x)$ for any $x \in \Sigma_1^*$. Moreover, 1-NLINSV is the subset of 1-NLINMV, containing only single-valued partial functions. In particular, 1-NLINMV$_t$ and 1-NLINSV$_t$ denote the collections of all *total* functions in 1-NLINMV and in 1-NLINSV, respectively. Clearly, 1-FLIN(partial) $\subseteq$ 1-NLINSV $\subsetneqq$ 1-NLINMV and 1-FLIN $\subseteq$ 1-NLINSV$_t$ $\subsetneqq$ 1-NLINMV$_t$. For simplicity, we concentrate only on length-preserving functions in this section.

Note that, for any function $f \in$ 1-NLINMV, we can nondeterministically decide whether x is in dom(f) and thus dom(f) belongs to the class 1-NLIN, which equals REG by Theorem 4.1.

The relationship between functions and languages is given as follows. Define $L[f] = \{[\begin{smallmatrix} x \\ y \end{smallmatrix}] \mid y \in f(x)\}$ for a multi-valued partial function f.

Lemma 4.7 *For any multi-valued partial function* $f \in$ LPF, $f \in$ 1-NLINMV *if and only if* $L[f]$ *is in* 1-DLIN.

Proof. Let f be any length-preserving multi-valued partial function. Assume that f is computed by a linear-time nondeterministic 1TM M. Consider the machine N that behaves as follows: on input $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$, nondeterministically compute z in $f(x)$ from x and check if $y = z$. This machine N places $L[f]$ in 1-NLIN, which equals 1-DLIN. Conversely, assume that $L[f]$ is recognized by a linear-time nondeterministic 1TM N. We define another machine M as follows: on input x guess $y \in \Sigma^n$ (by writing y in the second track), run N on input $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$, If N accepts, output y. Clearly, M computes f and thus f is in 1-NLINMV. □

Now, we wish to show the following major collapse result, which extends the collapse 1-NLIN = REG shown in Section 4.1.

Theorem 4.8 1-NLINSV $\cap$ LPF = 1-FLIN(partial) $\cap$ LPF *and thus* 1-NLINSV$_t$ $\cap$ LPF = 1-FLIN $\cap$ LPF.

Theorem 4.8 is a direct consequence of the following key lemma. We first introduce the notion of refinement. For any two multi-valued partial functions f and g from Σ_1^* to Σ_2^*, we say that f is a *refinement* of g if, for any $x \in \Sigma_1^*$, (i) $f(x) \subseteq g(x)$ (set inclusion) and (ii) $f(x) = \emptyset$ implies $g(x) = \emptyset$.

Lemma 4.9 *Every length-preserving* 1-NLINMV *function has a* 1-FLIN(partial) *refinement.*

The crucial part of the proof of Lemma 4.9 is the construction of a "folding machine" from a given nondeterministic 1TM. A folding machine rewrites the cells only in its input area, where the *input area* refers to the area of the tape cells in which the input symbols are initially written. For later use, we give a general description of a folding machine below.

Construction of a Folding Machine. Let $M = (Q, \Sigma, \Gamma, \delta, q_0', q_{acc}, q_{rej})$ be any 1TM (even a quantum 1TM in Section 8) that always halts in linear time. The *folding machine* N is constructed from M as follows. Choose the minimal integer k such that $\mathrm{Time}_M(x) \leq k|x|$ for all inputs x of length ≥ 3. Let q_0, q_1, q_2, q_3 be all new internal states not in Q. Moreover, we prepare new states of the form $[\begin{smallmatrix} i \\ q \end{smallmatrix}]$ for every number $i \in [-2k, 2k-1]_{\mathbb{Z}}$ and every state $q \in Q$. Let x be an arbitrary input given in the input tape of M.

1) The machine N starts in the initial state q_0. If the input x is empty, then N immediately enters M's halting state without moving its head. Hereafter, we assume that x is a nonempty string of the form $\sigma_1\sigma_2\cdots\sigma_n$, where each σ_i is a bit.

2) In this preprocessing phase, the machine N re-designs its tape as shown in Figure 1 by moving its head. The tape cells indexed between $-2k(|x|-1)$ and $2k(|x|-1)-1$ are partitioned into $4k$ blocks of $|x|-1$ cells. These blocks are indexed orderly from the leftmost block to the rightmost block using integers ranging from $-2k$ to $2k-1$. In particular, block 0 contains the input string $\sigma_1\sigma_2\cdots\sigma_{n-1}$. We split the tape into $4k$ tracks, which are indexed from the top to the bottom using $-2k$ to $2k-1$. The machine N first places ¢ in all tracks of odd indices and enters state q_1 in stepping right. The machine keeps moving its head rightward in state q_1. When the head scans the first blank symbol, if $|x| \geq 3$ then N enters state q_2 and steps back, and N enters M's halting state otherwise. In a single step, N places \$ in all tracks of even indices, shifts σ_n in track 0 to track 1,

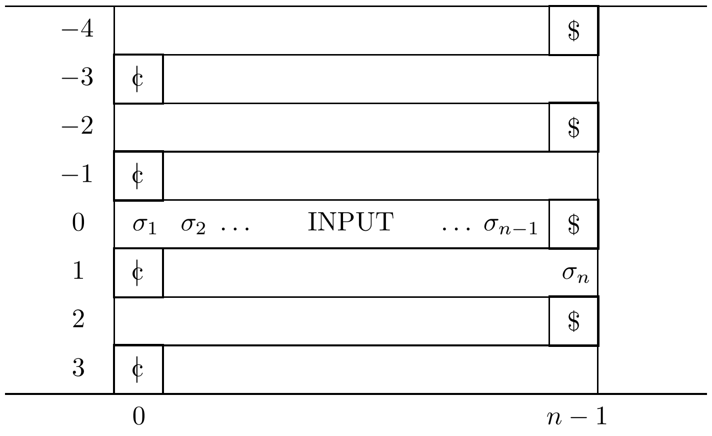

Figure 1: The track design of the tape of N where $k = 2$.

enters state q_3, and steps to the left. The head then returns to the start cell in state q_3. Note that this phase is done in a reversible fashion.

3) The machine N simulates M's move by folding M's tape content into $4k$ tracks of the input area. While M stays within block i in state q, N simulates M's move on track i with internal state $[\,{}^{i}_{q}\,]$. If i is even, then N moves its head in the same direction as M does. Otherwise, N moves the head in the opposite direction. In particular, at the time when M's head leaves the last (first, resp.) cell of block $2j$ to its adjacent block by rewriting symbol σ and entering state q, N instead enters state $[\,{}^{2j+1}_{q}\,]$ ($[\,{}^{2j-1}_{q}\,]$, resp.), writes symbol σ in track $2j$ ($2j$, resp.), and moves its head to the right (right, resp.). On the contrary, at the time when M's head leaves block $2j+1$, M moves the head similarly but in the opposite direction.

Note that N's head never visits outside of the input area. This simulation takes exactly the same amount of time as M's.

Consider the set S of all crossing sequences of N. For any two crossing sequences $v, v' \in S$ and any tape symbol σ, we write $v \to_\sigma v'$ if v is a crossing sequence of the left-boundary of σ and v' is a crossing sequence of the right-boundary of σ along a certain computation path of N on input $x\sigma y$ for certain strings x and y. For any computation path p of N on any nonempty input x, it is important to note that $v_0 = ()$, the *empty sequence*, and $v_f = (q_1, q_2)$ are respectively the unique crossing sequences at the left-boundary and the right boundary of x along computation path p. We can translate this computation path p on input $x = \sigma_1\sigma_2\cdots\sigma_n$ into its corresponding series of crossing sequences, $v_0, v_1, \ldots, v_n$, satisfying the following conditions: $v_n = v_f$ and $v_{i-1} \to_{\sigma_i} v_i$ for each $i \in [1,n]_{\mathbb{Z}}$.

Now, we return to the proof of Lemma 4.9.

Proof of Lemma 4.9. Let f be any length-preserving multi-valued partial function in 1-NLINMV. There exists a linear-time nondeterministic 1TM $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ that computes f. Consider the folding machine constructed from N. Because of the output convention of a one-tape machine, we modify this folding machine as follows. When M halts, N moves its head leftward. When N hits the left end-marker, it moves back the head by changing the tape symbol to the symbol written in the 0th track and the 1st track (which contains the original output symbols of M in the same arrangement as its input symbols).

When N hits the right end-marker, it steps right to the first blank symbol by entering a halting state (either q_{acc} or q_{rej}) of M. Note that this new machine also produces the outcome of M and enters exactly the same halting state of M. For convenience, we call this machine N as well.

Let CS be the set of all crossing sequences of M. Assume that all elements in CS are enumerated so that we can always find the minimal element in any subset of CS. Let $v_0 = ()$ and $v_f = (q_1, q_2)$ as before.

For any two elements $v, v' \in CS$ and any symbol σ with $v \to_\sigma v'$, define the output symbol $last(v, \sigma, v')$ written in the cell where σ is initially written. This symbol is easily deduced from (v, v', σ) since $v \to_\sigma v'$.

We define the deterministic 1TM $M = (Q', \Sigma', \Gamma', S_0, \delta', q_{acc}, q_{rej})$ as follows. Let x be an arbitrary input.

First phase) Let $S_0 = \{v_0\}$ and $S_f = \{v_f\}$. In this phase, all inner states are subsets of CS. Assume that we scan a symbol σ in a state S, which is a subset of CS. Let $V_\sigma(S) = \{v \in S \mid \exists\, v' \in CS\ [v \to_\sigma v']\}$ and $S' = \{v' \in CS \mid \exists\, v \in V_\sigma(S)\ [v \to_\sigma v']\}$. Note that (*) for any $v' \in S'$, there exists a $v \in V_\sigma(S)$ such that

$v \to_\sigma v'$. If S' is empty, then enter a new rejecting state. Otherwise, enter the state S' and change σ to $[\ {}^{\sigma}_{V_\sigma(S)}\ ]$. After reading x, when the head scans the first blank symbol, we change the blank symbol to $[\ {}^{b}_{S_f}\ ]$, unless $x \notin \mathrm{dom}(f)$, and step to the left by entering state v_f.

Second phase) The starting state of the second phase is v_f and the head scans the rightmost symbol of input x. Assume that we scan a symbol $[\ {}^{\sigma}_{V}\ ]$ in state v', where V is a subset of CS. If V is empty, then enter a rejecting state. Otherwise, choose the minimal element v in the set $\{w \in V \mid w \to_\sigma v'\}$. Note that if M passes the first phase and enters the second phase then V cannot be empty. It must have the form $V_\sigma(S)$ and v' must be in the set $S' = \{v' \in CS \mid \exists\, w \in V_\sigma(S)\ [w \to_\sigma v']\}$. Because of (*), we can always find the above v. Change the symbol $[\ {}^{\sigma}_{V}\ ]$ to $last(v, \sigma, v')$ and move the head to the left by entering state v. When the head reaches the blank symbol, we enter a new accepting state.

The above machine M produces at most one output string of $f(x)$ for each input x. If $x \notin \mathrm{dom}(f)$, all computation paths are rejecting paths and thus M never reaches an accepting state. Let g be the partial function computed by M. It is clear that g is a refinement of f. This completes the proof. $\Box$

Another consequence of Lemma 4.9 is the non-existence of one-way functions in 1-FLIN. A (total) function f is called *one-way* if (i) $f \in$ 1-FLIN and (ii) there is no function $g \in$ 1-FLIN such that $f(g(f(x))) = f(x)$ for all x. As a special case, f is a *one-way permutation* if (i) f is one-way in 1-FLIN mapping from Σ^* to Σ^* and (ii) for any $n \in \mathbb{N}$, f restricted to Σ^n is a bijection.

Corollary 4.10 *There is no length-preserving one-way function in* 1-FLIN. *Hence, no one-way permutation exists in* 1-FLIN.

Proof. Assume that a length-preserving one-way function f exists in 1-FLIN. Let f^{-1} be the multi-valued partial function defined by $f^{-1}(y) = \{x \mid f(x) = y\}$ for each y. Note that f^{-1} is length-preserving and belongs to 1-NLINMV. Lemma 4.9 ensures the existence of a 1-FLIN(partial) function g that is a refinement of f^{-1}. Consider the following 1TM M: on input y, check $y \in \mathrm{dom}(g)$. If not, output any fixed string of length $|x|$ (e.g., $0^{|x|}$). Otherwise, compute $g(y)$ and output it. Since $\mathrm{dom}(g)$ is in 1-DLIN, M can be deterministic. Clearly, this M inverts f (that is, $f(M(f(x))) = f(x)$ for all x). Thus, f is not one-way. $\Box$

5 Alternating Computation

Chandra, Kozen, and Stockmeyer [4] introduced the concept of alternating Turing machines as a natural extension of nondeterministic Turing machines. We first give a general description of an alternating 1TM. Let L be any set of labels, called a *label set*, which defines a type of alternation. An *alternating 1TM with label set* L is defined similar to a nondeterministic 1TM except that its internal states are all labeled with symbols in L (this labeling is done by the function g that maps the set of internal states to L). All the nodes of a computation tree are evaluated inductively as either T (true) or F (false) from the leaves to the root according to the label of an internal state given in each node. An alternating 1TM M *accepts* input x exactly when the root of the computation tree of M on x is evaluated T.

The k*-alternation* means that the number of the times when internal states change between different labels is at most $k-1$ along any computation path. Henceforth, we focus our interest on the label set $L = \{\exists, \forall\}$, where $\exists$ reads "existential" and $\forall$ reads "universal." For instance, a nondeterministic Turing machine can be viewed as an alternating Turing machine whose internal states are all labeled $\exists$. Let k and T be any functions from $\mathbb{N}$ to $\mathbb{N}$ satisfying that $k(n) \le T(n)$ for all $n \in \mathbb{N}$. The notation 1-$\Sigma_{k(n)}\mathrm{Time}(T(n))$ denotes the collection of all languages recognized by certain $T(n)$-time alternating 1TMs with *at most* $k(n)$-alternation starting with an $\exists$-state. In particular, we write 1-$\Sigma^{\mathrm{LIN}}_{k(n)}$ for 1-$\Sigma_{k(n)}\mathrm{Time}(O(n))$. To the contrary, 1-ALIN denotes the collection of all languages recognized by linear-time alternating 1TMs with unbounded alternation.

Of our particular interest are alternating 1TMs with a constant number of alternations. When k is a constant in $\mathbb{N}^+$, it clearly holds that REG $\subseteq$ 1-$\Sigma^{\mathrm{LIN}}_k \subseteq$ 1-ALIN. We further claim that 1-Σ^{LIN}_k collapses to REG. This generalizes the earlier collapse result REG = 1-NLIN.

Theorem 5.1 REG $= \bigcup_{k\in\mathbb{N}^+}$ 1-Σ^{LIN}_k.

Theorem 5.1 follows from Proposition 4.5 and the following lemma. In this lemma, we show that alternation can be viewed as an application of many-one 1-NLIN-reductions.

Lemma 5.2 *Let $k \in \mathbb{N}$. A set A is in 1-$\Sigma_{k+1}^{\mathrm{LIN}}$ if and only if A is many-one 1-NLIN-reducible to $\overline{B}$ for a certain set B in 1-$\Sigma_{k}^{\mathrm{LIN}}$.*

The proof of Theorem 5.1 proceeds by induction on k. The base case $k = 1$ is already shown in Theorem 4.1 since an alternating 1TM with 1-alternation starting with an $\exists$-state is identical to a nondeterministic 1TM. The induction step $k > 1$ is carried out as follows. Assume that A is in 1-$\Sigma_{k}^{\mathrm{LIN}}$. Lemma 5.2 yields the existence of a set $B \in$ 1-$\Sigma_{k-1}^{\mathrm{LIN}}$ satisfying $A \leq_{m}^{\text{1-NLIN}} \overline{B}$. By the induction hypothesis, B falls into REG and hence so does $\overline{B}$. Therefore, $A \in \text{1-NLIN}_{m}^{\overline{B}} \subseteq \text{1-NLIN}_{m}^{\mathrm{REG}} = \text{1-NLIN}$, which is REG.

Proof of Lemma 5.2. (Only If – part) For a given set A, take any linear-time alternating 1TM M with at most k-alternation that recognizes A over alphabet Σ_1. Without loss of generality, we can assume that M never visits the cell indexed -1 since, otherwise, we can "fold" a computation into two tracks, in which the first track simulates the tape region of nonnegative indices, and the second track simulates the tape region of negative indices.

First, we define the linear-time nondeterministic 1TM M' as follows. On input x, M' marks the start cell and then simulates M. Whenever M writes a blank symbol, replace it with a new symbol not in Σ_1, say #. If M enters the first $\forall$-state p, M' marks the cell (by changing its tape symbol a to the new symbol $[\,{}^{a}_{p}\,]$), moves the head back to the start cell, and erases the mark. It is important to note that M' has at least $|x|$ non-blank symbols in its tape when it halts. Let Σ consist of all symbols of the form $[\,{}^{a}_{p}\,]$ for any symbol $a \in \Sigma_1$ and any $\forall$-state p.

Next, we define N as follows. Let $\Sigma_2 = \Sigma_1 \cup \Sigma \cup \{\#\}$. On input y in Σ_2^*, N changes all #s to the blank symbol, finds in the tape the first symbol from Σ, say $[\,{}^{a}_{p}\,]$, and changes it back to a. Recover the state p as well. By starting with this $\forall$-state p, N simulates M. The desired set B consists of all input strings rejected by N. Obviously, B is in 1-$\Sigma_{k-1}^{\mathrm{LIN}}$. It is also not difficult to show that A is many-one 1-NLIN-reducible to $\overline{B}$ via M'.

(If – part) The proof is done by induction on $k \in \mathbb{N}$. Assume that A is many-one 1-NLIN-reducible to $\overline{B}$ via a reduction machine N, where B is in 1-$\Sigma_{k}^{\mathrm{LIN}}$. Choose a linear-time alternating 1TM M that recognizes B with k-alternation starting with an $\exists$-state. Define $\overline{M}$ to be the one obtained from M by exchanging $\forall$-states and $\exists$-states and swapping an accepting state and a rejecting state. Now, we define N' as follows: on input x, run N, and, when it halts, run $\overline{M}$. Clearly, N' runs in linear time. It is also easy to show that N' recognizes A with $(k+1)$-alternation. Thus, A belongs to 1-$\Sigma_{k+1}^{\mathrm{LIN}}$. □

The argument of the proof of Theorem 5.1 does not rule out the possibility of REG $\neq$ 1-ALIN.

6 Probabilistic Computation

Probabilistic (or randomized) computation has been proven to be essential to many applications in computer science. Probabilistic extensions of deterministic Turing machines have been studied since as early as the 1950s. For simplicity, we use Gill's notion of probabilistic Turing machines with *fair coin* flips [12]. Formally, a *probabilistic 1TM* is defined as a nondeterministic 1TM that has *at most* two nondeterministic choices at each step, which is often called a *coin toss* (or *coin flip*) if there are exactly two choices. Each coin toss is made with probability 1/2. Instead of taking an expected running time, we define a probabilistic 1TM M to be *$T(n)$-time bounded* if, for every x, any computation path of M on input x has length at most $T(|x|)$. The probability associated with computation path s equals 2^{-m}, where m is the number of coin tosses along path s. The *acceptance probability* of M on input x, denoted $p_M(x)$, is the sum of all probabilities of any accepting computation paths. We say that *M recognizes L with error probability at most ε* if, for every x, (i) if $x \in L$, then $1 - p_M(x) \leq \varepsilon$; and (ii) if $x \notin L$, then $p_M(x) \leq \varepsilon$.

We begin with a key lemma, which is a probabilistic version of Lemma 4.3. Kaneps and Freivalds [19], following Rabin's [28] result, proved a similar result for probabilistic finite automata.

Lemma 6.1 *Let L be any language and let M be any probabilistic 1TM that recognizes L with error probability at most $\varepsilon(n)$, where $0 \leq \varepsilon(n) < 1/2$ for all numbers $n \in \mathbb{N}$. For each $n \in \mathbb{N}$, let S_n be the set of all crossing sequences at any critical-boundary of x along any accepting computation path of M on every input x of length $\leq n$. Then, $N_L(n) \leq 2^{|S_n|\lceil |S_n|/\delta(n)\rceil}$ for all $n \in \mathbb{N}$, where $\delta(n) = 1/2 - \varepsilon(n)$.*

Proof. Fix $n \in \mathbb{N}$ arbitrarily. For every string $x \in \Sigma^{\leq n}$ and every crossing sequence $v \in S_n$, let $w_l(x|v)$

($w_r(v|z)$, resp.) be the sum, over all z (all x, resp.) with $|xz| \leq n$, of all probabilities of the coin tosses made during the head staying in the left-side region of the right-boundary of x (the right-side region of the left-boundary of z, resp.) along any accepting computation path of M on input xz. By their definitions, we have $0 \leq w_l(x|v), w_r(v|z) \leq 1$. The key observation is that the acceptance probability of M on input xz with $|xz| \leq n$ equals $\sum_{v\in S_n} w_l(x|v)w_r(v|z)$.

Now, we say that x *n-supports* (i, v) if $|x| \leq n$, $i \in [0, \lceil |S_n|/\delta(n)\rceil - 1]_{\mathbb{Z}}$, $v \in S_n$, and $i \cdot \delta(n)/|S_n| \leq w_l(x|v) \leq (i+1)\delta(n)/|S_n|$. Define the set $\mathrm{Supp}_n(x) = \{(i, v) \mid x\ n\text{-supports } (i, v)\}$. We first show that, for every x, y, z with $|xz| \leq n$ and $|yz| \leq n$, if $xz \in L$ and $\mathrm{Supp}_n(x) = \mathrm{Supp}_n(y)$, then $yz \in L$. This is shown as follows. Since $\mathrm{Supp}_n(x) = \mathrm{Supp}_n(y)$, $|w_l(x|v) - w_l(y|v)| \leq \delta(n)/|S_n|$ for all $v \in S_n$. Thus, $|p_M(xz) - p_M(yz)| = \left|\sum_{v\in S_n}(w_l(x|v) - w_l(y|v)) \cdot w_r(v|z)\right| \leq \sum_{v\in S_n} |w_l(x|v) - w_l(y|v)| \leq \sum_{v\in S_n} \delta(n)/|S_n| = \delta(n)$. Since $xz \in L$, we obtain $p_M(xz) \geq 1 - \varepsilon(n)$, which yields $p_M(yz) > \varepsilon(n)$. Therefore, $yz \in L$.

Note that $N_L(n)$ is bounded above by the number of distinct $\mathrm{Supp}_n(x)$'s for all $x \in \Sigma^{\leq n}$. Therefore, $N_L(n)$ is at most $2^{|S_n|\lceil |S_n|/\delta(n)\rceil}$. □

We first focus on the case where the error probability of a probabilistic 1TM is bounded away from 1/2. For any language L and any probabilistic 1TM M, we say that M *recognizes L with bounded-error probability* if there exists a constant $\varepsilon > 0$ such that M recognizes L with error probability at most $\frac{1}{2} - \varepsilon$. We define 1-BPTime($T(n)$) as the collection of all languages recognized by $T(n)$-time probabilistic 1TM with bounded error probability. We also define 1-BPTime($\mathcal{T}$) similar to the nondeterministic case. The *one-tape bounded-error probabilistic linear-time* class 1-BPLIN is 1-BPTime($O(n)$).

For any language L recognized by a probabilistic 1TM M with bounded-error probability in time $o(n \log n)$, by Lemma 4.2, the number of all crossing sequences of M is upper-bounded by a certain constant independent of input. From Lemma 6.1, it follows that $N_L(n)$ is bounded above by a function of the error bound $\epsilon(n)$. Since $\epsilon(n)$ is bounded away from 1/2, we obtain $N_L(n) = O(1)$, which yields the regularity of L. Therefore, the following theorem holds with the help of Proposition 3.1.

Theorem 6.2 REG = 1-BPTime($o(n \log n)$) $\neq$ 1-BPTime($O(n \log n)$).

Leaving from bounded-error probabilistic computation, we hereafter focus on unbounded-error probabilistic computation. We define 1-PLIN to be the collection of all languages of the form $\{x \in \Sigma^* \mid p_M(x) > 1/2\}$ for certain linear-time probabilistic 1TMs M. Different from 1-BPLIN, 1-PLIN does not collapse to REG because the non-regular set $L_> = \{a^m b^n \mid m > n\}$ is in 1-PLIN.

The following theorem establishes an automaton-characterization of 1-PLIN. We write 1-synPLIN for the subset of 1-PLIN defined by linear-time probabilistic 1TMs that are particularly *synchronous*.

Theorem 6.3 1-PLIN = 1-synPLIN = SL$_{rat}$.

The class 1-PLIN is easily shown to be closed under complementation and symmetric difference, where the *symmetric difference* between two sets A and B is $(A - B) \cup (B - A)$. These properties also result from Theorem 6.3 using the corresponding properties of SL$_{rat}$.

Theorem 6.3 follows from two key lemmas: Lemmas 6.4 and 6.5. We begin with Lemma 6.4 whose proof is based on the simulation.

Lemma 6.4 SL$_{rat}$ $\subseteq$ 1-synPLIN.

Proof. Let L be any language recognized by a rational 1PFA $N = (S, \Sigma, \pi, \{T(\sigma) \mid \sigma \in \Sigma\}, \eta)$ with a rational cut point. Without loss of generality, we can assume that (i) $L = L(N, 1/2)$, (ii) $S = [1, s]_{\mathbb{Z}}$ for a certain number $s \in \mathbb{N}^+$, (iii) an entry of π equals 1, and (iv) there is a positive integer m satisfying the following property: for any $\sigma \in \Sigma$ and any $i, j \in S$, the (i, j)-entry of $T(\sigma)$, denoted $T(\sigma)_{i,j}$, is of the form $r_{i,j}(\sigma)/2^m$ for a certain number $r_{i,j}(\sigma) \in \mathbb{N}$. Write F for the set of all final states of N.

Our goal is to construct a synchronous probabilistic 1TM M that simulates N in linear time. The desired machine M works as follows. In case where the input is the empty string, M immediately enters q_{acc} or q_{rej} depending on $\lambda \in L$ or $\lambda \notin L$, respectively. Henceforth, assuming that our input is not empty, we give an algorithmic description of N's behavior. First, we repeat phases 1)-2) until M finishes scanning all input symbols.

1) In scanning a symbol σ, M first generates 2^m branches by tossing exactly m fair coins without moving its head. All the branches are used for the simulation of a single step of N in phase 2).

2) In a single step, M simulates N's move in the following manner. Assume that N is in state i. Note that,

for any $j \in S$, N changes state i to state j with the transition probability $T(\sigma)_{i,j}$ $(= r_{i,j}(\sigma)/2^m)$ while scanning symbol σ. To simulate such a transition, we choose exactly $r_{i,j}(\sigma)$ branches out of the 2^m branches and follow the transition from state i to state j. More precisely, along the ℓ-th branch generated by the coin tosses made in phase 1), M simulates N's transition from state i to state j if $\sum_{k=1}^{j-1} r_{i,k}(\sigma) < \ell \leq \sum_{k=1}^{j} r_{i,k}(\sigma)$. We then force M's head to move to the right-adjoint cell.

3) When M finishes reading the entire input, its head must sit in the first blank cell. If N reaches a finial state in F, then M enters q_{acc}; otherwise, M enters q_{rej}. This completes the description of M.

By our simulation, the acceptance probability of M on input x coincides with that of N on the same input. Moreover, our simulation makes M's computation paths terminate all at once since we toss the equal number of coins. Therefore, L is in 1-synPLIN via M. □

The following lemma complements Lemma 6.4.

Lemma 6.5 *For any probabilistic 1TM M running in linear time, there exists a rational 1GPFA N such that $p_N(x) = p_M(x)$ for any input x.*

To lead to the desired consequence 1-PLIN $\subseteq$ SL_{rat}, take any language L in 1-PLIN and consider a linear-time probabilistic 1TM M that recognizes L with unbounded-error probability. Lemma 6.5 guarantees the existence of a rational 1GPFA N for which $L = L(N, 1/2)$. Hence, L is in GSL_{rat}, which is known to equal SL_{rat}.

Proof of Lemma 6.5. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ be any linear-time probabilistic 1TM. In this proof, we need the folding machine M' constructed from M. To simplify the proof, we further modify M' as follows. When M' halts in a certain halting state, we force its head to move rightward and cross the left-boundary of the original input by entering the same halting state. Note that the accepting probability of this modified machine is the same as that of M. For notational convenience, we use the notation M to denote this modified machine. For this M, the crossing sequence at the left-boundary of any input is $v_0 = ()$ and the crossing sequence at the right-boundary of any input is $v_f = (q_1, q_2, q_{acc})$ on any accepting computation path.

We wish to construct a rational 1GPFA $N = (S, \Sigma, \pi, \{T(\sigma) \mid \sigma \in \Sigma\}, \eta)$ satisfying that $p_M(x) = \pi\, T(x)\, \eta$ for all inputs $x \in \Sigma^*$. The desired automaton N is defined in the following manner. Let S denote the set of all crossing sequences of M. It follows from Lemma 4.2 that S is a finite set. Let σ be an arbitrary symbol in Σ. For any pair (u, v) of elements in S, we define $P(u;\sigma;v)$ to be the *probability* of the following event $\mathcal{E}$.

> Event $\mathcal{E}$: Consider any computation of M where M starts on input $y\sigma z$ with the head initially scanning the left-most symbol of the input $y\sigma z$ for a certain pair (y, z) of strings. In a certain computation path of M, (i) u coincides with the crossing sequence at the left-boundary of σ, (ii) v is the crossing sequence at the right-boundary of σ, and (iii) $u \to_\sigma v$.

Clearly, $P(u;\sigma;v)$ is a rational number since M flips only fair coins. Let $x = \sigma_1 \cdots \sigma_n$ be any nonempty input string, where each σ_i is in Σ. By the correspondence between a series of crossing sequences and a computation path, the acceptance probability $p_M(x)$ equals $\sum_{\vec{v}} \prod_{i=1}^{n} P(v_{i-1};\sigma_i;v_i)$, where the sum is taken over all sequences $\vec{v} = (v_0, v_1, \ldots, v_n)$ from S with $v_n = v_f$. For each $\sigma \in \Sigma$, let $T(\sigma)$ be the $|S| \times |S|$ matrix whose (u, v)-element is $P(u;\sigma;v)$ for any $u, v \in S$. The row vector π has 1 or 0 in the vth column if $v = v_0$ or $v \neq v_0$, respectively. Letting $F = \{v_0, v_f\}$ if $\lambda \in L$ and $F = \{v_f\}$ otherwise, we define η to be the column vector whose vth component is 1 (0, resp.) if $v \in F$ ($v \notin F$, resp.).

By the above definition, it is not difficult to verify that $p_N(x) = \pi\, T(x)\, \eta = p_M(x)$ for any input x, as required. □

Dwork and Stockmeyer [11] placed the language $L_{cen} = \{x1y \mid x, y \in \{0,1\}^*, |x| = |y|\}$ in the difference CFL $-$ SL_{rat}. Note that L_{cen} also falls into REG/n. Later, Macarie [23] showed the proper containment $\mathrm{SL}_{rat} \subsetneqq \mathrm{L}$, where L is the class of all languages recognized by multiple-tape deterministic Turing machines, with a read-only input tape and read/write work-tapes, that uses $O(\log n)$ tape-space on all tapes except for the input tape. We thus conclude the following corollary of Theorem 6.3.

Corollary 6.6 CFL $\cap$ REG/$n \nsubseteq$ 1-PLIN *and* 1-PLIN $\subsetneqq$ L.

7 Counting Computation

Counting issues naturally arise in many fields of computer science. For instance, a decision problem of determining whether there exists a "solution" induces the problem of counting the number of such solutions. In this way, Valiant [35] considered the notion of counting Turing machines to study the complexity of counting. We investigate counting functions computed by one-tape linear-time counting Turing machines.

7.1 Counting Functions

A *counting 1TM* is a variant of a nondeterministic 1TM, which behaves like a nondeterministic 1TM except that, when it halts, it outputs the number of all accepting computation paths. Let $\#M(x)$ denote the outcome of a counting 1TM M on input x.

Similar to the function class #P [35], we use the notation 1-#LIN (pronounced "one sharp lin") for the collection of all total functions f, from Σ^* to $\mathbb{N}$, which are computed by certain linear-time counting 1TMs. This class 1-#LIN naturally includes 1-FLIN by identifying any natural number n with the nth string over alphabet Σ (in the standard order) and by constructing a linear-time nondeterministic 1TM which branches n computation paths, given the nth string as an input. Another useful function class is 1-GapLIN, which is defined as the class of all total functions whose values are the difference between the number of accepting paths and the number of rejecting paths of linear-time nondeterministic 1TMs. Such functions are conventionally called *gap functions*.

We can prove the following closure property. For convenience, write 1-NLINMV$_{dis}$ for the collection of all partial multi-valued functions computed by certain nondeterministic 1TMs whose valid computation paths always output distinct values.

Lemma 7.1 *For any functions f, g in* 1-GapLIN *and any function h in* 1-NLINMV$_{dis}$*, the following functions all belong to* 1-GapLIN*: $f \cdot g$, $f + g$, $f - g$, and $\lambda x. \sum_{y \in h(x)} f([\begin{smallmatrix} x \\ y \end{smallmatrix}])$.*

Proof. We show the lemma only for the last function. For any given functions $f \in$ 1-GapLIN and $h \in$ 1-NLINMV, let $k(x) = \sum_{y \in h(x)} f([\begin{smallmatrix} x \\ y \end{smallmatrix}])$ for any string x. Take a nondeterministic 1TM M_h computing h in linear time and also a linear-time counting 1TM M_f that computes f. Consider the following counting 1TM N. On input x, N produces $[\begin{smallmatrix} x \\ x \end{smallmatrix}]$ in the tape and runs M_h using only the lower track. When M_h halts, by our output convention, it leaves $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$, where $y \in h(x)$, in the tape. Next, N runs M_f on this $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$. It follows that, for every x, $\#N(x)$ equals $\sum_{y \in h(x)} \#M_f([\begin{smallmatrix} x \\ y \end{smallmatrix}])$, which is exactly $k(x)$. Hence, k is in 1-GapLIN. □

The above closure property implies that, for instance, 1-GapLIN = 1-#LIN − 1-#LIN, where the notation $\mathcal{F} - \mathcal{G}$ stands for the set $\{f - g \mid f \in \mathcal{F}, g \in \mathcal{G}\}$.

Lemma 7.2 1-GapLIN = 1-#LIN − 1-#LIN.

Proof. Let h be any function in 1-#LIN − 1-#LIN. Take two functions $f, g \in$ 1-#LIN satisfying $h = f - g$. Since $f, g \in$ 1-GapLIN, the difference function $f - g$ is also in 1-GapLIN by the closure property of 1-GapLIN. Hence, h is in 1-GapLIN.

Conversely, let h be any function in 1-GapLIN. There exists a linear-time counting 1TM M that witnesses h; that is, $h(x) = \#M(x) - \#\overline{M}(x)$ for all x, where $\#\overline{M}(x)$ denotes the number of all rejecting computation paths of M on input x. Define $f(x) = \#M(x)$ and $g(x) = \#\overline{M}(x)$ for every x. Clearly, f is in 1-#LIN. It is also easy to show that g is in 1-#LIN. Since $h = f - g$, h belongs to 1-#LIN − 1-#LIN. □

Lemma 7.2 implies that 1-GapLIN is a proper subset of 1GAF$_{rat}$.

Lemma 7.3 1-GapLIN $\subsetneqq$ 1GAF$_{rat}$.

Proof. The inequality is trivial since functions in 1GAF$_{rat}$ possibly output non-integer values whereas 1-GapLIN contains only integer-valued functions. The remaining inclusion relationship is proven in two steps. Firstly, the proof technique of Lemma 6.5 allows us to prove the containment 1-#LIN $\subseteq$ 1GAF$_{rat}$ by defining $P(u; \sigma; v)$ to be the number of accepting computation paths instead of probabilities. Secondly, the lemma is immediate from Lemma 7.2 and the following closure property of 1GAF$_{rat}$: for any $f_1, f_2 \in$ 1GAF$_{rat}$ and any constants $c_1, c_2 \in \mathbb{Q}$, the function $c_1 f_1 + c_2 f_2$ is in 1GAF$_{rat}$ [24], where $(c_1 f_1 + c_2 f_2)(x) = c_1 f_1(x) + c_2 f_2(x)$. □

Theorem 6.3 and Lemma 7.3 build a bridge between counting computation and unbounded-error probabilistic computation. Here, we show that 1-PLIN can be characterized in terms of 1-GapLIN.

Proposition 7.4 1-PLIN $= \{A \mid \exists f \in$ 1-GapLIN $[A = \{x \mid f(x) > 0\}]\}$.

Proof. Assume that $A = \{x \mid f(x) > 0\}$ for a certain function f in 1-GapLIN. Lemma 7.3 puts f into $1\mathrm{GAF}_{rat}$. This makes A belong to GSL_{rat}. Since $\mathrm{GSL}_{rat} = \mathrm{SL}_{rat}$, Lemma 6.4 ensures that A is indeed in 1-PLIN.

Conversely, let A be any set in 1-PLIN. By Lemma 6.5, A is also in SL_{rat}. By the proof of Lemma 6.4, we can recognize A in linear time by a synchronous probabilistic 1TM M which tosses the equal number of coins on all computation paths on each input. Let N be the machine obtained from M by exchanging the role of q_{acc} and q_{rej}. Now, we view M and N as counting 1TMs. Define f and g be the functions computed by the counting machines M and N, respectively. It follows from the definition that, for every x, $x \in A$ if and only if $f(x) > g(x)$ if and only if $(f-g)(x) > 0$. Hence, $A = \{x \mid (f-g)(x) > 0\}$. Since $f-g$ is in 1-GapLIN, this completes the proof. □

We already know the inclusion 1-FLIN $\subseteq$ 1-#LIN. Furthermore, Proposition 7.4 yields the separation between 1-FLIN and 1-#LIN.

Corollary 7.5 1-FLIN $\neq$ 1-#LIN.

Proof. We want to show that if 1-FLIN = 1-#LIN then 1-DLIN = 1-PLIN. Since REG $\neq$ 1-PLIN, it immediately follows that 1-FLIN $\neq$ 1-#LIN. Now, assume that 1-FLIN = 1-#LIN. Let A be any set in 1-PLIN and we wish to show that A is also in 1-DLIN. By Lemma 7.2 and Proposition 7.4, there exist two functions f and g in 1-#LIN such that $A = \{x \mid f(x) > g(x)\}$. By our assumption, these functions fall into 1-FLIN. Using deterministic 1TMs that compute f and g in linear time, we can produce $[\ {}^{f(x)}_{g(x)}\ ]$ in binary in linear time. We can further determine whether $f(x) > g(x)$ by comparing $f(x)$ and $g(x)$ bitwise. Thus, A belongs to 1-DLIN. Therefore, 1-DLIN = 1-PLIN, as required. □

7.2 Counting Complexity Classes of Languages

The function classes 1-#LIN and 1-GapLIN naturally introduce several useful counting complexity classes. First, we define the counting class 1-SPLIN to be the collection of all languages whose characteristic functions belong to 1-GapLIN. Furthermore, let 1-$\oplus$LIN (pronounced "one parity lin") consist of all languages of the form $\{x \in \Sigma^* \mid f(x) \equiv 1 \mod 2\}$ for certain functions f in 1-#LIN. Obviously, REG $\subseteq$ 1-SPLIN $\subseteq$ 1-$\oplus$LIN. Despite their complex definitions, these counting classes are no more powerful than REG. Hereafter, we prove the collapse of these classes down to REG. Our proof uses a crossing sequence argument.

Theorem 7.6 REG = 1-SPLIN = 1-$\oplus$LIN.

Proof. It suffices to show that 1-$\oplus$LIN $\subseteq$ REG. This can be shown by modifying the proof of Lemma 6.1. Here, we define $w_l(x|v)$ and $w_r(v|z)$ to denote the number of accepting computation paths instead of probabilities. Recall that $\#M(u)$ denotes the outcome (that is, the number of accepting paths) of M on input u. Note that, for every x, z with $|xz| \leq n$, $\#M(xz) = \sum_{v \in S_n} w_l(x|v) w_r(v|z)$.

We also define $\mathrm{Supp}_n(x)$ to be the set $\{v \in S_n \mid w_l(x|v) \equiv 0 \mod 2\}$. We show that, for every x, y, z with $|xz| \leq n$ and $|yz| \leq n$, if $xz \in L$ and $\mathrm{Supp}_n(x) = \mathrm{Supp}_n(y)$ then $yz \in L$. This is shown as follows. Since $\mathrm{Supp}_n(x) = \mathrm{Supp}_n(y)$, $S_n - \mathrm{Supp}_n(x)$ equals $S_n - \mathrm{Supp}_n(y)$. Thus, $\#M(xz) - \#M(yz) = \sum_{v \in S_n} (w_l(x|v) - w_l(y|v)) \cdot w_r(v|z)$, which equals $\sum_{v \in \mathrm{Supp}_n(x)} (w_l(x|v) - w_l(y|v)) \cdot w_r(v|z) + \sum_{v \in S_n - \mathrm{Supp}_n(x)} (w_l(x|v) - w_l(y|v)) \cdot w_r(v|z)$. The first sum is even since each $w_l(x|v) - w_l(y|v)$ is even. We show that the second sum is also even. For any $v \in S_n - \mathrm{Supp}_n(x)$, $w_l(x|v)$ and $w_l(y|v))$ are both odd. Hence, $w_l(x|v) - w_l(y|v)$ should be even. Therefore, $\#M(xz) - \#M(yz) \equiv 0 \mod 2$. Since $\#M(xz) \equiv 0 \mod 2$, we have $\#M(yz) \equiv 0 \mod 2$. This implies, $yz \in L$.

Note that $N_L(n)$ is bounded above by the number of distinct $\mathrm{Supp}_n(x)$'s for all $x \in \Sigma^{\leq n}$. Therefore, $N_L(n)$ is bounded above a certain term of $|S_n|$, which is clearly a constant. Therefore, L is in REG. □

We show an immediate consequence of Theorem 7.6 regarding low sets. Note that the notion of many-one 1-NLIN reducibility can be further expanded into other complexity classes (such a complexity class is called *many-one relativizable*). We can naturally define the many-one relativized counting class 1-$\#\mathrm{LIN}_m^A$ relative to oracle A as the collection of all single-valued total functions f such that there exists a linear-time nondeterministic 1TM M satisfying the following: on any input x, M produces an output y_p along each computation path p and $f(x)$ equals the number of such computational paths p for which $y_p \in A$. Similarly, the many-one relativized class 1-GapLIN$_m^A$ is defined using the difference $|\{p \mid y_p \in A\}| - |\{p \mid y_p \notin A\}|$. A language A is called *many-one low* for a relativizable complexity class $\mathcal{C}$ of languages or of functions if $\mathcal{C}_m^A \subseteq \mathcal{C}$. A class $\mathcal{D}$ is *many-one low* for $\mathcal{C}$ if every set in $\mathcal{D}$ is many-one low for $\mathcal{C}$. We use the notation low$_m\mathcal{C}$ to denote the collection of all languages that are many-one low for $\mathcal{C}$. For instance, low$_m$1-NLIN = REG since 1-NLIN$_m^{\mathrm{REG}}$ = 1-NLIN. The following corollary is a direct consequence of Theorem 7.6.

Corollary 7.7 REG = low$_m$1-#LIN = low$_m$1-GapLIN.

Proof. To prove the corollary, we first note that REG $\subseteq$ low$_m$1-#LIN since 1-#LIN$_m^{\mathrm{REG}}$ = 1-#LIN. Conversely, for any set A in low$_m$1-GapLIN, since $\chi_A \in$ 1-#LIN$_m^A \subseteq$ 1-GapLIN$_m^A$, it follows that $\chi_A \in$ 1-GapLIN. Thus, A is in 1-SPLIN, which equals REG by Theorem 7.6. Therefore, low$_m$1-GapLIN $\subseteq$ REG. □

We further introduce another counting class 1-C$_=$LIN (pronounced "one C equal lin") as the collection of all languages of the form $\{x \mid f(x) = 0\}$ for certain functions f in 1-GapLIN. This class 1-C$_=$LIN properly contains REG because the non-regular language $L_{eq} = \{a^n b^n | n \geq 0\}$ clearly belongs to 1-C$_=$LIN. Using the closure property of 1-GapLIN, we can easily show that 1-C$_=$LIN is closed under intersection and union. This is shown as follows. Let $A = \{x \mid f(x) = 0\}$ and $B = \{x \mid g(x) = 0\}$ for certain functions $f, g \in$ 1-GapLIN. Obviously, $A \cap B = \{x \mid f^2(x) + g^2(x) = 0\}$ and $A \cup B = \{x \mid f(x)g(x) = 0\}$. By Lemma 7.1, $A \cap B$ and $A \cup B$ are in 1-C$_=$LIN.

Proposition 7.4 leads to the following relationship between 1-C$_=$LIN and 1-PLIN.

Lemma 7.8 1-C$_=$LIN $\subseteq$ 1-PLIN $\subseteq$ 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$.

Lemma 7.8 implies that 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$ = 1-NLIN$_m^{\text{1-PLIN}}$. This is seen as follows. The inclusion 1-NLIN$_m^{\text{1-C}_=\text{LIN}} \subseteq$ 1-NLIN$_m^{\text{1-PLIN}}$ is obvious. Since 1-PLIN $\subseteq$ 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$, it follows that 1-NLIN$_m^{\text{1-PLIN}} \subseteq$ 1-NLIN$_m^{\text{1-NLIN}_m^{\text{1-C}_=\text{LIN}}}$. Thus the class 1-NLIN$_m^{\text{1-NLIN}_m^{\text{1-C}_=\text{LIN}}}$ coincides with 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$ by Proposition 4.5.

Proof of Lemma 7.8. Using Lemma 7.1 and Proposition 7.4, we can easily show that 1-C$_=$LIN $\subseteq$ 1-PLIN. Next, we want to show that 1-PLIN $\subseteq$ 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$. Assume that A is any set in 1-PLIN. Again by Proposition 7.4, choose a gap function $f \in$ 1-GapLIN satisfying that $A = \{x \mid f(x) > 0\}$. To simplify our proof, we assume that the empty string λ is not in A. Let $N = (Q, \Sigma, \Gamma, q_0, q_{acc}, q_{rej})$ be any linear-time counting 1TM that witnesses f. Without loss of generality, we can assume that (i) at each step, N makes at most two nondeterministic choices and (ii) $f(x) \neq 0$ for all strings x in Σ^*. Since N runs in linear time, let k be the minimal positive integer such that $\mathrm{Time}_N(x) < k|x|$ for any nonempty string x. It thus follows that $0 \leq f(x) < 2^{k|x|}$ for any string x.

For brevity, write Δ_k for the set $\{0,1\}^k$ and assume the standard lexicographic order on Δ_k. Now, we define the reduction machine M as follows: on input x, guess a string over Δ_k, say s, of length $|x|$ and produce $[\begin{smallmatrix} x \\ s \end{smallmatrix}]$. Note that the total number of such $[\begin{smallmatrix} x \\ s \end{smallmatrix}]$'s is exactly $2^{k|x|}$. We then define the nondeterministic 1TM N' as follows: on input $[\begin{smallmatrix} x \\ s \end{smallmatrix}]$, guess a string s' in $(\Delta_k)^{|x|}$ in the third track. If $x = \lambda$, then accept the input immediately and halt. Assume otherwise that $x \neq \lambda$. If $s < s'$, then produce both an accepting path and a rejecting path. Such an s does not contribute to the gap function witnessed by N'. Consider the case where $s = s'$. In this case, simulate N on input x. At length, when $s > s'$, reject the input and halt.

Let g be the gap function induced by N'. For any nonempty string x and any string $s \in (\Delta_k)^{|x|}$, we obtain $g([\begin{smallmatrix} x \\ s \end{smallmatrix}]) = f(x) - (i_s - 1)$, provided that s is the lexicographically i_sth string in $(\Delta_k)^{|x|}$. Define $B = \{x \mid g(x) = 0\}$. Clearly, B is in 1-C$_=$LIN. Note that if $f(x) > 0$ then $g([\begin{smallmatrix} x \\ s \end{smallmatrix}]) = 0$ for the $f(x) + 1$st string s. Otherwise, since $f(x) < 0$, $g([\begin{smallmatrix} x \\ s \end{smallmatrix}])$ has a negative value. Hence, A is many-one 1-NLIN-reducible to B via M; namely, A is in 1-NLIN$_m^B$, which is a subset of 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$. □

The class 1-synC$_=$LIN is the subset of 1-C$_=$LIN defined only by linear-time synchronous counting 1TMs. A similar argument to the proof of Lemma 6.5 yields the containment 1-C$_=$LIN $\subseteq$ SL$_{rat}^=$. Moreover, similar to Lemma 6.4, we can prove that SL$_{rat}^= \subseteq$ 1-synC$_=$LIN. Therefore, we obtain the following theorem.

Theorem 7.9 1-C$_=$LIN = 1-synC$_=$LIN = SL$_{rat}^=$

Turakainen [31] proved that SL$_{rat}^=$, co-SL$_{rat}^=$, and SL$_{rat}$ are all different classes. From Theorem 7.9, we immediately obtain the following corollary.

Corollary 7.10 1-C$_=$LIN $\neq$ co-1-C$_=$LIN $\neq$ 1-PLIN.

Recall the language L_{num} defined in Section 3. While Lemma 3.3 places L_{num} outside of REG/n, L_{num} obviously belongs to 1-C$_=$LIN. We can also prove that the non-context-free language $L_{abc} = \{a^n b^n c^n \mid n \in \mathbb{N}^+\}$ is in 1-C$_=$LIN $\cap$ REG/n since $L_1 = \{a^n b^n c^m \mid m, n \in \mathbb{N}^+\}$ and $L_2 = \{a^m b^n c^n \mid m, n \in \mathbb{N}^+\}$ are clearly in 1-C$_=$LIN and their intersection $L_{abc} = L_1 \cap L_2$ is also in 1-C$_=$LIN. Overall, we have the following separation.

Proposition 7.11 1-C$_=$LIN $\nsubseteq$ REG/n *and* 1-C$_=$LIN $\cap$ REG/n $\nsubseteq$ CFL.

Finally, the complexity class that is many-one low for 1-C$_=$LIN, denoted low$_m$1-C$_=$LIN, is located between REG and 1-C$_=$LIN. This is proven similar to the proof of Corollary 7.7. There is no known separation between low$_m$1-C$_=$LIN and 1-C$_=$LIN.

8 Quantum Computation

The notion of a quantum Turing machine was introduced by Deutsch [7] in mid 1980s and later reformulated by Bernstein and Vazirani [3] to model a quantum computation. In accordance with our definition of 1TMs, we use a slightly more general model of one-tape quantum Turing machines that allow their tape head to stay still [37, 38].

A *(measure-once) one-tape quantum Turing machine* (abbreviated 1QTM) is similar to the classical 1TM $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ except that its transition function δ is a map from $Q\times\Gamma$ to the vector space $\mathbb{C}^{Q\times\Gamma\times\{L,N,R\}}$. The *configuration space* of M is the Hilbert space spanned by the set of all configurations of M as the computational basis. An element of this configuration space is called a *superposition of configurations*. A 1QTM M is *well-formed* if its time-evolution operator preserves the ℓ_2-norm (i.e., Euclidean norm), where the *time-evolution operator* for M is the operator that maps a superposition of configurations to a superposition of the configurations resulted by an application of the quantum transition function δ of M. For any subset K of $\mathbb{C}$, a 1QTM is said to have *K-amplitudes* if all amplitudes in δ are drawn from K. By ignoring its nonzero transition amplitudes, δ can be seen as a nondeterministic transition function. For clarity, we use the notation $\hat{\delta}$ to express this nondeterministic transition function. Similar to the classical case, a *(classical) computation path* of a 1QTM is defined as a series of configurations, each of which is obtained from its previous configuration by an application of $\hat{\delta}$. These classical computation paths form a *classical computation tree*. Any quantum computation can be viewed as its corresponding classical computation tree in which each edge is weighted by its associated nonzero amplitude.

Unlike classical Turing machines, there is a subtle but arguable issue concerning the definition of the halting condition of a QTM. In accordance to the classical halting condition, we define the *running time* of a 1QTM M on input x as the minimal nonnegative integer t such that, in the classical computation tree T representing the computation of M on input x, all configurations in T at time t become halting configurations for the first time. If such a t exists, we say that *M halts*[‡] *at time t*. This halting condition makes any time-bounded 1QTM be viewed as classical "synchronous" machines. A time-bounded 1QTM M is called *well-behaved* if, when M halts, the tape head halts in the same cell (not necessarily the start cell) in all halting configurations of the classical computation tree representing the computation of M. Moreover, M is *stationary* if it is well-behaved and the head always halts in the start cell.

The *acceptance probability* of a 1QTM M on input x, denoted $p_M(x)$, is the sum of all the squared magnitudes of accepting configurations (i.e., a configuration with the state q_{acc}) in any superposition generated at the time when M halts on input x. Let K be any nonempty subset of $\mathbb{C}$. We introduce the *one-tape bounded-error quantum linear-time* class 1-BQLIN$_K$ as the collection of all languages L that satisfy the following condition: there exist a linear-time well-formed stationary 1QTM M with K-amplitudes and an error bound $\varepsilon > 0$ such that, for every string x, (i) if $x \in L$, then $p_M(x) \geq 1/2 + \varepsilon$ and (ii) if $x \notin L$, then $p_M(x) \leq 1/2 - \varepsilon$.

It is important to note that our linear-time 1QTMs may not simulate linear-time 2-way quantum finite automata given in [21] mainly because of the synchronous condition of our 1QTMs. On the contrary, the synchronous condition enables us to prove a strong connection between 1QTMs and 1-GapLIN in Lemma 8.1.

We prove a key lemma, which shows how to compute the acceptance probability of a 1QTM with $\mathbb{Q}$-amplitudes. The lemma has a similar flavor to Theorem 3(4) in [38]. In the following proof, we use the folding machine obtained from a given 1QTM.

[‡]Originally, Bernstein and Vazirani [3] defined a quantum Turing machine to "halt" at time t if the superposition at time t consists of all halting configurations and, at less than t, the superposition contains no halting configuration. If a QTM runs in polynomial time, our definition is fundamentally the same in power as that of Bernstein and Vazirani. See [37, 38].

Lemma 8.1 *Let M be any well-formed stationary 1QTM with $\mathbb{Q}$-amplitudes. If M always halts, then there exist a constant $d \in \mathbb{N}^+$ and a function f in* 1-GapLIN *such that $p_M(x) = f(x) \cdot d^{-\mathrm{Time}_M(x)}$ for every string x.*

Proof. Given a 1QTM M, we work on its folding machine $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$, which simulates M in N's tape using only the input area. Notice that N may no longer well-formed. Since N uses rational amplitudes, we can choose the minimal integer $c \in \mathbb{N}^+$ satisfying that every amplitude of N has the form r/c, where r is a certain integer. Fix x arbitrarily and let y be any computation path of N on input x. When N halts, an accepting configuration of N depends only on the tape content because N's inner state and its head position in the accepting configuration are predetermined. It thus suffices to consider a final tape content of N. Let z be any final tape content of N. Note that $|z| = |x|$ since N rewrites cells only in the input area. We denote by $amp_N(x, y, z)$ the amplitude associated with the accepting computation path y of N on input x leading to the final tape content z. Since N is synchronous, this value $amp_N(x, y, z) \cdot c^{\mathrm{Time}_M(x)}$ is always an integer.

Now, we define the function f_+ as $f_+([\begin{smallmatrix} x \\ z \end{smallmatrix}]) = c^{\mathrm{Time}_M(x)} \cdot \sum_y amp_N(x, y, z)$, where the sum $\sum$ is taken over all accepting computation paths y of N on input x that leads to the final tape content z with *positive* amplitude. We translate the 1QTM N into the classical nondeterministic 1TM $\hat{N}$ in such a way that, whenever N makes a transition with a transition amplitude of the form m/c for certain integers m, c with $c > 0$, $\hat{N}$ produces $|m|$ nondeterministic branches. As a result, for each of such y's, we can generate exactly $amp_N(x, y, z) \cdot c^{\mathrm{Time}_M(x)}$ branches that all reach accepting configurations. To determine the sign of the amplitude associated with each computation path of N, we further prepare two sets of inner states for $\hat{N}$ and move one set to another whenever the amplitude sign changes by an application of δ. Therefore, f_+ is a function in 1-#LIN. Similarly, we define $f_-([\begin{smallmatrix} x \\ z \end{smallmatrix}]) = c^{\mathrm{Time}_M(x)} \cdot \sum'_y amp_N(x, y, z)$, where the sum $\sum'$ is taken over all accepting computation paths of N on input x that leads to the final tape content z with *negative* amplitude. We also obtain $f_- \in$ 1-#LIN.

Recall that the acceptance probability $p_M(x)$ equals the sum of $(\sum_y amp_N(x, y, z) - \sum'_y amp_N(x, y, z))^2$ over all possible final tape contents z of N. In other words, $c^{2\mathrm{Time}_M(x)} \cdot p_M(x) = \sum_{z \in \Gamma^{|x|}} (f_+([\begin{smallmatrix} x \\ z \end{smallmatrix}]) - f_-([\begin{smallmatrix} x \\ z \end{smallmatrix}]))^2$. From the closure property of 1-GapLIN (Lemma 7.1), the function appearing in the right-hand side of the last equation clearly belongs to 1-GapLIN. For the desired constant d, set $d = c^2$. This completes the proof. □

By a simple application of Lemma 8.1, we show the following.

Proposition 8.2 REG $\subseteq$ 1-BQLIN$_{\mathbb{Q}}$ $\subseteq$ 1-PLIN.

Proof. Note that every (deterministic) reversible 1TM can be viewed as a well-formed stationary 1QTM with $\mathbb{Q}$-amplitudes. This implies that 1-revDLIN $\subseteq$ 1-BQLIN$_{\mathbb{Q}}$. Proposition 3.2 therefore implies REG $\subseteq$ 1-BQLIN$_{\mathbb{Q}}$.

We want to show the second inclusion. Let L be any set in 1-BQLIN$_{\mathbb{Q}}$. There exists a linear-time well-formed 1QTM M that recognizes L with bounded-error probability. Moreover, M uses only rational amplitudes. By amplifying the success probability, we can assume without loss of generality that, for every x, either $p_M(x) \geq 2/3$ or $p_M(x) \leq 1/3$. By Lemma 8.1, we find a constant $d \in \mathbb{N}^+$ and a function $f \in$ 1-GapLIN such that $f(x) = p_M(x) \cdot d^{\mathrm{Time}_M(x)}$ for every input x. Now, we define $g(x) = d^{\mathrm{Time}_M(x)}$ for each x. It thus follows that $x \in L$ implies $3f(x) > 2g(x)$ and $x \notin L$ implies $3f(x) < g(x)$. To complete the proof, we need to define $h(x) = 3f(x) - 2g(x)$, which is also in 1-GapLIN by Lemma 7.1. This h satisfies $L = \{x \mid h(x) > 0\}$. Hence, L is in 1-PLIN. □

A variant of quantum Turing machine, so-called a "nondeterministic" quantum Turing machine, which is considered as a quantum analogue of a nondeterministic Turing machine, was introduced by Adleman et al. [1]. Let K be any nonempty subset of $\mathbb{C}$. A language L is in 1-NQLIN$_K$ if there exist a linear-time well-formed stationary 1QTM M with K-amplitudes such that, for every x, $x \in L$ if and only if M accepts input x with positive probability.

We show that 1-NQLIN$_{\mathbb{Q}}$ can be precisely characterized by linear-time counting 1TMs.

Proposition 8.3 1-NQLIN$_{\{0,\pm 3/4,\pm 4/5,\pm 1\}}$ = 1-NQLIN$_{\mathbb{Q}}$ = co-1-C$_=$LIN.

Proposition 8.3 is obtained by combining two lemmas: Lemmas 8.4 and 8.5.

Lemma 8.4 1-NQLIN$_{\mathbb{Q}}$ $\subseteq$ co-1-C$_=$LIN.

Proof. Let L be any language in 1-NQLIN$_{\mathbb{Q}}$. Choose a linear-time well-formed stationary 1QTM M satisfying that $L = \{x \mid p_M(x) > 0\}$. Applying Lemma 8.1, we obtain a constant $d \in \mathbb{N}^+$ and a function f in 1-GapLIN such that $f(x) = p_M(x) \cdot d^{\mathrm{Time}_M(x)}$ for any string x. It immediately follows that, for every x, $x \in L$ if and only if $f(x) \neq 0$. Therefore, L belongs to the complement of 1-C$_=$LIN. □

Finally, we prove the remaining inclusion co-1-C$_=$LIN $\subseteq$ 1-NQLIN$_{\{0,\pm 3/5,\pm 4/5,\pm 1\}}$. From the fact 1-C$_=$LIN $=$ SL$^{=}_{rat}$, it suffices to show that co-SL$^{=}_{rat}$ $\subseteq$ 1-NQLIN$_{\{0,\pm 3/4,\pm 4/5,\pm 1\}}$. In the proof of Lemma 8.5, we use the following two unitary transformations acting on $\{|s\rangle \mid s \in [0,3]_{\mathbb{Z}}\}$. Let $U|0\rangle = \frac{3}{5}|0\rangle + \frac{4}{5}|1\rangle$, $U|1\rangle = -\frac{4}{5}|0\rangle + \frac{3}{5}|1\rangle$, and $U|s\rangle = |s\rangle$ for $s \in \{2,3\}$. Let $V|s\rangle = \frac{(1-s)4}{5}|s\rangle + \frac{3}{5}|(s+2) \mod 4\rangle$ for $s \in \{0,2\}$ and $V|s\rangle = \frac{3}{5}|s\rangle + \frac{(s-2)4}{5}|(s+2) \mod 4\rangle$ for $s \in \{1,3\}$.

Lemma 8.5 co-SL$^{=}_{rat}$ $\subseteq$ 1-NQLIN$_{\{0,\pm 3/4,\pm 4/5,\pm 1\}}$.

Proof. Let L be any set in co-SL$^{=}_{rat}$. There exists a rational 1PFA $N = (S, \Sigma, \pi, \{T(\sigma) \mid \sigma \in \Sigma\}, \eta)$ such that $\overline{L} = L^{=}(N, \epsilon)$ for a rational cut point ϵ. Similar to the proof of Lemma 6.4, we can assume that (i) $L = \{x \in \Sigma^* \mid p_N(x) \neq 1/2\}$, (ii) $S = [1,\ell]_{\mathbb{Z}}$ for a certain number $\ell \in \mathbb{N}^+$, (iii) an component of π is 1, and (iv) there is a positive integer m satisfying the following property: for any $\sigma \in \Sigma$ and any $i,j \in S$, $T(\sigma)_{i,j}$ is of the form $r_{i,j}(\sigma)/2^m$ for a certain number $r_{i,j}(\sigma) \in \mathbb{N}$. Let F be the set of all final states of N.

In what follows, we wish to construct a linear-time well-formed stationary 1QTM M with $\{0,\pm 3/5,\pm 4/5,\pm 1\}$-amplitudes such that $p_M(x) > 0$ iff $p_N(x) \neq 1/2$ for any nonempty string x. This implies that L is in 1-NQLIN$_{\mathbb{Q}}$. Initially, assume that M is in the initial state q_0 scanning the start cell and M's tape contains an input $x = \sigma_1 \ldots \sigma_n$, where each $\sigma_j \in \Sigma$ and $n \geq 1$. For simplicity, let $\Delta = \{0,1\}^m$ be an alphabet. In a preprocessing phase, M replaces each input symbol σ by a new symbol $\left[\begin{smallmatrix}\sigma\\ 0^m\end{smallmatrix}\right]$, $0^m \in \Delta$, and returns its head to the start cell. We focus only on the content of the cells of indices between 0 and n.

1) The machine M simulates a series of "coin flips" of N by generating a certain superposition of configurations. By moving the head rightward, M applies the transformation $U^{\otimes m}$ to the symbol 0^m given in $\left[\begin{smallmatrix}\sigma\\ 0^m\end{smallmatrix}\right]$. When M reaches the first blank symbol in the nth cell, it returns the head to the start cell. Next, M simulates the computation of N by applying repeatedly the following transition rule: $\delta(p_a, \left[\begin{smallmatrix}\sigma\\ k\end{smallmatrix}\right]) = |p_b\rangle|\left[\begin{smallmatrix}\tau_a\\ k\end{smallmatrix}\right]\rangle|R\rangle$ if $\sum_{i=1}^{b-1} r_{a,i}(\sigma) \leq k < \sum_{i=1}^{b} r_{a,i}(\sigma)$ for any $a,b \in S$, $\sigma \in \Sigma$, and $k \in \Delta$.

2) When M reaches the n-th cell in state p_a for a certain $a \in S$, M writes down the symbol 0 or 1 if $a \in F$ or $a \notin F$, respectively, and then M enters a new state s_a. Let $r_{\vec{k}}$ be the symbol written in the nth cell when $\vec{k}$ is written in the lower track. Note that $p_N(x) = |\{\vec{k} \in \Delta^n \mid r_{\vec{k}} = 0\}| \cdot 2^{-mn}$.

3) This phase is a so-called "uncomputing." First, M recovers the original tape symbols. By moving the head leftward, we repeatedly apply the following transition rule: $\delta(s_a, \left[\begin{smallmatrix}\tau_b\\ k\end{smallmatrix}\right]) = (s_b, \left[\begin{smallmatrix}\sigma\\ k\end{smallmatrix}\right], L)$ if $\sum_{i=1}^{a-1} r_{b,i}(\sigma) \leq k < \sum_{i=1}^{a} r_{b,i}(\sigma)$. While moving the head rightward, M applies $U^{\otimes m}$ to the symbol k_j in $\left[\begin{smallmatrix}\sigma_j\\ k_j\end{smallmatrix}\right]$ just as in phase 1). At this point, the amplitude associated with then tape content $\left[\begin{smallmatrix}\sigma_1 & \cdots & \sigma_n\\ 1^m & \cdots & 1^m\end{smallmatrix}\right] r_{\vec{k}}$ is exactly $(\frac{12}{25})^{mn}$.

4) We need to make the accepting paths and rejecting paths of N interfere to each other. This is done by applying the unitary transformation $W = UV$ to the nth cell since W maps $|0\rangle$ and $|1\rangle$ to $|0\rangle$ with amplitude $12/25$ and $-12/25$, respectively. The machine M then dumps its inner state into the blank cell and enters a new inner state q_{check}.

5) The machine N checks if the content of the cells of indices between 0 and n is exactly $\left[\begin{smallmatrix}\sigma_1 & \cdots & \sigma_n\\ 1^m & \cdots & 1^m\end{smallmatrix}\right] 0$. If so, M enters q_{acc} and otherwise M enters q_{rej}. This phase is clearly done in a reversible fashion. A simple calculation shows that $p_M(x) = \left(\frac{12}{25}\right)^{2mn} \left|\sum_{\vec{k} \in \Delta^n} W_{0,r_{\vec{k}}}\right|^2$, which equals $\left(\frac{24}{25}\right)^{2mn+2} (p_N(x) - 1/2)^2$. This completes the description of M.

It immediately follows that, for every nonempty string x, $p_M(x) > 0$ if and only if $p_N(x) \neq 1/2$, as required. □

9 Epilogue

By exploring the relationship to formal language theory as well as automata theory, we have studied the computational complexity of one-tape linear-time Turing machines of various machine types. Since these machines are relatively weak, we have proven the collapses and separations of several complexity classes without any unproven assumptions. Hennie's crossing sequence arguments and various simulation techniques are proven to be viable tools throughout this paper. Nonetheless, we have left numerous questions unsolved. Challenging these questions may bring in new proof techniques.

For future research on the theory of one-tape linear-time Turing machines, we suggest five important future directions of research.

i) The model of Turing machines has significantly evolved over specially the past three decades. We have shown in this paper that different machine types significantly alter the power of computation. Other types of Turing machines that we have not discussed in this paper include metric Turing machines, bottleneck Turing machines, and interactive Turing machines (see, e.g., [22, 8, 13]). We need to explore the computational power of such models and study the properties of complexity classes induced in terms of these models.

ii) Despite the ability to alter the tape content, we have shown that many Turing machines working as language recognizers cannot be more powerful than their corresponding finite automata. To study the power of Turing machines, we need to explore their special ability to compute "functions." It has been long time that functions have been studied in terms of search problems, optimization problems, and approximation problems. The study of such functions may present different perspectives to our understandings of one-tape computation.

iii) It is natural to ask what is the most complex language existing in a given complexity class. The theory of NP-completeness, for instance, sheds light on this question using various polynomial-time reductions. On the contrary, most one-tape liner-time complexity classes we have studied in this paper are unlikely to possess complete problems via many-one 1-DLIN-reductions. It there any weak reducibility that sheds light on the relative complexity of languages.

iv) We have considered advised computations; however, the role of advice has not fully studied in this paper. It is important to investigate how much extra power advice can give to an underlying computation. Moreover, advised computations are often characterized by non-uniform computations. We also need to study the non-uniformity of one-tape linear-time computations in connection to advice.

v) Relativization has had a great success in the polynomial-time complexity theory. Throughout this paper, we have studied only many-one relativization since many-one relativization is of the simplest form. The investigation of other meaningful relativizations is also necessary for one-tape linear-time complexity classes.

We hope that the exploration of the above structural complexity issues on resource-bounded computations leads to the better understandings of the effect of bounded resources.

Acknowledgments. The first author is grateful to the Japan Science and Technology Corporation and the 21st Century COE Security Program of Chuo University for financial support. He also thanks Masahiro Hachimori for valuable suggestions on probabilistic computations. The second author thanks Harumichi Nishimura and Raymond H. Putra for fruitful discussions on Turing machines.

References

[1] L. M. Adleman, J. DeMarrais, and M. A. Huang. Quantum computability. *SIAM J. Comput.*, **26** (1997), 1524–1540.

[2] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Develop.*, **17** (1973), 525–532.

[3] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Comput.*, **26** (1997), 1411–1473.

[4] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *J. ACM* **28** (1981), 114–133.

[5] A. Condon. Bounded error probabilistic finite state automata, in *Handbook of Randomized Computing* (eds, Sanguthevar Rajasekaran, Panos M. Pardalos, John H. Reif, and José D. P. Rolim), Kluwer, 2001.

[6] C. Damm and M. Holzer. Automata that take advice. *Proc. 20th Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science Vol.969, pp.149–158, Springer, 1995.

[7] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. Lond.* A, **400** (1985), pp.97–117.

[8] D. Du and K. Ko. *Theory of Computational Complexity.* John Wiley & Sons. 2000.

[9] P. Dúriś and Z. Galil. Two tapes are better than one for nondeterministic machines. *SIAM J. Comput.* **13** (1984), 219–227.

[10] C. Dwork and L. J. Stockmeyer. A time complexity gap for two-way probabilistic finite state automata. *SIAM J. Comput.*, **19** (1990), 1011–1023.

[11] C. Dwork and L. Stockmeyer. Finite state verifiers I: The power of interaction. *J. ACM*, **39** (1992), 800–828.

[12] J. Gill, Computational complexity of probabilistic Turing machines. *SIAM J. Comput.*, **6** (1977), 675–695.

[13] L. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion*, Springer, 2002.

[14] F. C. Hennie. One-tape, off-line Turing machine computations. *Inform. Control*, **8** (1965), 553–578.

[15] J. E. Hopcroft and J. D. Ullman. *Formal Languages and Their Relation to Automata.* Addison-Wesley, 1969.

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

[17] R. M. Karp. Some bounds on the storage requirements of sequential machines and Turing machines. *J. ACM*, **14** (1967), 478–489.

[18] R. M. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathematique*, **28** (1982), 191–209.

[19] J. Kaneps and R. Freivalds. Minimal nontrivial space complexity of probabilistic one-way Turing machines. *Proc. 19th Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science Vol.452, pp.355–361, Springer, 1990.

[20] K. Kobayashi. On the structure of one-tape nondeterministic Turing machine time hierarchy. *Theor. Comput. Sci.*, **40** (1985), 175–193.

[21] A. Kondacs and J. Watrous. On the power of quantum finite state automata. *Proc. 38th IEEE Symposium on Foundations of Computer Science*, pp.66–75, 1997.

[22] M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences* **36** (1988), 490–509.

[23] I. I. Macarie. Space-efficient deterministic simulation of probabilistic automata. *SIAM J. Comput.*, **27** (1998), 448–465.

[24] I. I. Macarie. Closure properties of stochastic languages. Technical Report TR441. Department of Computer Science, University of Rochester, 1993.

[25] P. Michel. An NP-complete language accepted in linear time by a one-tape Turing machine. *Theor. Comput. Sci.*, **85** (1991), 205–212.

[26] W. Paul, N. Pippenger, E. Szemeredi, and W. Trotter. On determinism versus non-determinism and related problems. In *Proc. 24th IEEE Symposium on Foundations of Computer Science*, pp.429–438, 1983.

[27] E. Post. Finite combinatory process–formulation I. *J. Symbolic Logic*, **1** (1936), 103–105.

[28] M. O. Rabin. Probabilistic automata. *Inform. Control*, **6** (1963), 230–245.

[29] M. O. Rabin. Real time computation. *Israel Journal of Mathematics*, **1** (1963), 203–211.

[30] P. Turakainen. On stochastic languages. *Inform. Control*, **12** (1968), 304–313.

[31] P. Turakainen. On languages representable in rational probabilistic automata. *Annales Academiae Scientiarum Fennicae*, Ser.A **439** (1969), 4–10.

[32] P. Turakainen. Generalized automata and stochastic languages. *Proc. Amer. Math. Soc.*, **21** (1969), 303–309.

[33] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Society*, Ser.2, **42** (1936), 230–265.

[34] A. Turing. Rectification to 'On computable numbers, with an application to the Entscheidungsproblem.' *Roc. London Math. Society*, Ser.2, **43** (1937), 544–546.

[35] L. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.* **8** (1979), 189–201.

[36] T. Yamakami. Average case complexity theory. Ph.D. dissertation, University of Toronto, 1997. Technical Report 307/97, University of Toronto. See also ECCC Thesis Listings.

[37] T. Yamakami. A foundation of programming a multi-tape quantum Turing machine. *Proc. 24th Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science Vol.1672, pp.430–441, Springer, 1999.

[38] T. Yamakami. Analysis of quantum functions. *International Journal of Foundations of Computer Science*, **14** (2003), 815–852. A preliminary version appeared in the *Proc. 19th Conference on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, Vol.1738, pp.407-419, Springer, 1999.

[39] T. Yamakami and A. C. Yao. $\mathrm{NQP}_{\mathbb{C}} = \mathrm{co\text{-}C_{=}P}$. *Information Processing Letters*, **71** (1999), 63–69.

[40] A. C. Yao, Quantum circuit complexity, in *Proc. 34th Annual IEEE Symposium on Foundations of Computer Science*, pp.352–361, 1993.

Appendix

We show the proofs of Lemmas 4.2 and 4.3 for completeness.

Proof of Lemma 4.2. Let M be given as in the lemma and let q be the number of states of M. By its definition, M has at least three states (that is, $q \geq 3$). We define the function f from $\mathbb{N} - \{0,1\}$ to $\mathbb{R}^{\geq 0}$ by $f(n) = \sqrt{n \log n / T(n)}$. Since $T(n) = o(n \log n)$, $\lim_{n\to\infty} f(n) = \infty$. Choose the smallest number $c \in \mathbb{N}$ such that, for every $n \geq 2$, $\frac{3(q^{\frac{\log n}{f(n)}+1}-1)}{q-1} \leq n\left(1 - \frac{1}{f(n)}\right) + \frac{c \cdot f(n)}{\log n}$. Such a c exists since $q^{\frac{\log n}{f(n)}+1} = o(n)$.

Assume to the contrary that there exist a crossing sequence γ of length longer than c and input x ($|x| \geq 2$) such that γ is a crossing sequence at some critical-boundary b of x along some accepting computation path s of M on x. Such a crossing sequence is called *long*, and the other crossing sequences are called *short*.

Let x_0 be lexicographically the first input string that has a long crossing sequence. Let $n_0 = |x_0|$. Let s_0 be the shortest accepting computation path of M on input x_0 that generates a long crossing sequence. Note that $|s_0| \leq T(|x_0|)$ by our assumption. Moreover, let b_0 be the leftmost intercell boundary in the tape that corresponds to a certain long crossing sequence (say, γ_0) in path s_0.

Consider all critical-boundaries of x_0 along path s_0 that have crossing sequences of lengths $\leq \log n_0 / f(n_0)$. Let h be the number of all such critical-boundaries. Since the total computation steps in s_0 is equal to the sum of the lengths of any crossing sequences of every intercell boundary, we have $T(n_0) > c + (n_0 - h)\frac{\log n_0}{f(n_0)}$. The inequality comes from the assumption that the length of γ_0 is longer than c. Thus, we have:

$$\frac{h}{3} > \frac{1}{3}\left(n_0 - \frac{n_0}{f(n_0)} + \frac{c \cdot f(n_0)}{\log n_0}\right) \geq \frac{q^{\frac{\log n_0}{f(n_0)}+1} - 1}{q-1} \geq \sum_{i=0}^{\lfloor \log n_0 / f(n_0) \rfloor} q^i,$$

which is equal to the number of all crossing sequences of lengths $\leq \log n_0 / f(n_0)$. Hence there exist at least four distinct critical-boundaries b_1, b_2, b_3, b_4 that have an identical crossing sequence in path s_0. Thus at least two of them (say, b_1 and b_2) are on the same side of b_0.

Now, we delete the region between b_1 and b_2 from the tape. Let x_0' be the input string obtained from x_0 by this deletion. Clearly, $|x_0'| < |x_0|$. Moreover, the new path obtained from s_0 by this deletion is an accepting computation path of M on input x_0' and still has a crossing sequence whose length is greater than c. This contradicts the minimality of x_0. This completes the proof. □

Proof of Lemma 4.3. Let n be any integer in $\mathbb{N}$. For each $x \in \Sigma^{\leq n}$ and each $v \in S_n$, we say that x *n-supports* v if there exists a string z such that (i) $|xz| \leq n$, (ii) $xz \in L$, and (iii) v is the crossing sequence at an intercell boundary between x and z along a certain *accepting* computation path of M on input xz. For each $x \in \Sigma^{\leq n}$, let $\mathrm{Supp}_n(x) = \{v \in S_n \mid x \text{ } n\text{-supports } v\}$.

We first show that, for every strings x, y, z, if $|xz| \leq n$, $|yz| \leq n$, $xz \in L$, and $\mathrm{Supp}_n(x) = \mathrm{Supp}_n(y)$, then $yz \in L$. This is shown as follows. Assume that $xz \in L$. Let v be a crossing sequence between x and z along an accepting computation path of M on input xz. Clearly, $v \in \mathrm{Supp}_n(x)$. Since $\mathrm{Supp}_n(x) = \mathrm{Supp}_n(y)$, there exists a string z' such that v is a crossing sequence between y and z' along an accepting computation path of M on yz'. Assume that the tape head halts in the left region of v. Consider any computation of M on input yz. By the nature of crossing sequences, yz has an accepting computation. Thus $yz \in L$. Similarly, we obtain the same conclusion in the case where the tape head halts in the right region of v.

Note that $N_L(n)$ is bounded above by the number of distinct $\mathrm{Supp}_n(x)$'s for all $x \in \Sigma^{\leq n}$. Therefore, $N_L(n)$ is at most $2^{|S_n|}$. □