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 (one-head) linear-time Turing machines is essentially different from its polynomial-time counterpart since these 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 length of any longest computation path. We explore structural properties of one-tape linear-time Turing machines and clarify how the machines' resources affect their computational patterns and power.

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

1 Prologue

Computer science has revolved around the study of computation incorporated with the analysis and development of fast and efficient algorithms. The notion of a *Turing machine*, proposed by Turing [41, 42] and independently by Post [34] in the mid 1930s, is now regarded as a mathematical model of many existing computers. This machine model has long been a foundation of extensive studies in computational complexity theory. Early research unearthed the significance of various restrictions on the *resources* of machines: for instance, the number of work tapes, the number of heads, execution time bounds, memory space bounds, and machine types in use. This paper aims at the better understanding of how various resource restrictions directly affect the patterns and the power of computations.

The number of work tapes and also machine types of time-bounded Turing machines significantly alter their computational power. For instance, two-tape Turing machines are shown to be more powerful than any one-tape Turing machines [12, 35]. Even on the model of multiple-tape Turing machines, Paul, Pippenger, Szemeredi, and Trotter [32] proved in the early 1980s that linear-time nondeterministic Turing machines are more powerful than their deterministic counterparts.

Of particular interest in this paper is the model of one-tape (or single-tape) one-head linear-time Turing machines, apart from well-studied polynomial-time machines. Not surprisingly, this rather simple model proves a close tie to finite state automata. Despite its simplicity, such a model still offers complex structures. As a result, a theory of one-tape (one-head) linear-time complexity draws a picture quite different from multiple-tape models as well as polynomial-time models. It is thus possible for us to prove, for instance, the collapses and separations of numerous one-tape linear-time complexity classes without any unproven assumption, such as the existence of one-way functions.

Hennie [18] 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 deterministic finite state automata. 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 state changes is known as a *crossing sequence* generated at this boundary. Using this technical tool, 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

[*] An earlier version 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 in 2001.

[‡] Present address: School of Computer Science and Engineering, University of Aizu, 90 Kami-Iawase, Tsuruga, Ikki-machi, Aizu-Wakamatsu, Fukushima 965-8580, Japan.

is short, then this machine recognizes only a regular language. Using the *non-regularity* measure of Dwork and Stockmeyer [13], the second claim asserts that any language accepted by a machine with short crossing sequences has constantly-bounded non-regularity. Extending Hennie's argument, Kobayashi [25] later showed that any language recognized by one-tape $o(n\log n)$-time deterministic Turing machines should be regular as well. This time bound $o(n\log n)$ is actually 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 [44]. By taking his *weak definition* that defines the running time of a nondeterministic Turing machine to be the length of a "shortest" accepting path, Michel [30] demonstrated that one-tape nondeterministic Turing machines running in linear time (in the sense of his weak definition) solve even NP-complete problems. Clearly, his weak definition gives an enormous power to one-tape nondeterministic machines and therefore it does not seem to offer any interesting features of time-bounded nondeterminism. On the contrary, the *strong definition* (in Michel's term) requires the running time to be the length of any "longest" (both accepting and rejecting) computation path. This strong definition provides us with a reasonable basis to study the effect of linear-time bounded computations. We therefore adopt his strong definition of running time and, throughout this paper, all one-tape time-bounded Turing machines are assumed to accommodate this strong definition. By expanding Kobayashi's result, we prove that one-tape $o(n\log n)$-time nondeterministic Turing machines recognize only regular languages.

The model of *alternating Turing machines* of Chandra, Kozen, and Stockmeyer [6] naturally expand the model of nondeterministic machines. The number of alternations of such an alternating Turing machine seems to enhance the computational power of the machine; however, our strong definition of running time makes it possible for us to prove that a constant number of alternations do not give any additional computational power to one-tape linear-time alternating Turing machines; namely, such machines recognize only regular languages.

Apart from nondeterminism, *probabilistic Turing machines* with fair coin tosses of Gill [16], can present distinctive features. Any language recognized by a certain one-head one-way probabilistic finite automaton with unbounded-error probability is known as a *stochastic* language [35]. By employing a crossing sequence argument, we can show that any language recognized by one-tape linear-time probabilistic Turing machines with unbounded-error probability is just stochastic. This collapse result again proves a close relationship between one-tape linear-time Turing machines and finite state automata.

The model of Turing machines, nonetheless, presents distinguishing 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 and producing output strings (which are sometimes viewed as numbers). Such functions also serve as *many-one reductions* between two languages. To explore the structure of language classes, we introduce various types of "many-one one-tape linear-time" reductions. Nondeterministic many-one reducibility, for instance, plays an important role in showing the aforementioned 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. Intuitively, a one-way function is a function that is easy to compute but hard to invert. Restricted to one-tape linear-time deterministic computation, we can show that no one-way function exists.

The number of accepting computation paths of a time-bounded nondeterministic Turing machine has been a crucial player in computational complexity theory. With the notion of *counting Turing machines*, Valiant [43] initiated a systematic study in the late 1970s on the structural properties of counting such numbers. Counting Turing machines have been since then used to study the complexity of "counting" on numerous issues in computer science. The functions computed by these machines are called *counting functions* and complexity classes of languages defined in terms of such counting functions are generally referred to as *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 family 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 [9] and later reformulated by Bernstein and Vazirani [5]. Of all the known types of quantum Turing machines, we study only the following two machine types: bounded-error quantum Turing machines [5] 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 supplemental mechanism called *advice* to enhance the computational power of Turing machines. Karp and Lipton [24] formalized the notion of advice, which means additional information supplied to underlying computation besides an original input. We adapt their notion in our setting of one-tape Turing machines as well as finite state automata. We can demonstrate the existence of context-free languages that cannot be recognized by any one-tape linear-time deterministic Turing machines with advice.

2 Fundamental Models of Computation

This paper uses a standard definition of a Turing machine (see, e.g., [11, 20]) as a computational model. Of special interest are one-tape one-head Turing machines of various machine types. Here, we give brief descriptions of 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 $n \leq m$, an integer interval $[n, m]_{\mathbb{Z}}$ means 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 and Σ^* denotes the collection of all finite strings over Σ. Note that the *empty string* over any alphabet is always denoted λ. Let $\Sigma^{+} = \Sigma^* - \{\lambda\}$. For any 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 the difference $\Sigma^* - A$, and it is often denoted $\overline{A}$ if Σ is clear from the context. For any complexity class $\mathcal{C}$, the *complement* of $\mathcal{C}$, denoted co-$\mathcal{C}$, 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 another 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. Whenever f is single-valued, we write "$f(x) = y$" instead of "$y \in f(x)$" by identifying the set $\{y\}$ with y itself. Notice that *total* functions are also partial functions. 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 $\mathbb{N}$.

Let us give the basic definition of one-tape (one-head) Turing machines. A *one-tape (one-head) 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 will define different types of transition functions δ, which give rise to various types of 1TMs. 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 1TMs. A 1TM is said to be *synchronous* if all computation paths terminate at the same time on each input; namely, all the computation paths have the same length.

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 any longest computation path (no matter what halting state the machine reaches) of M on x. We often use the notation $T(n)$ to denote a time-bounding function of a given 1TM that maps $\mathbb{N}$ 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 upper-bounded 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 $a, b \in \Sigma$, $[\,{}^{a}_{b}\,]$ 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 $[\,{}^{x}_{y}\,]$ to denote the concatenation $[\,{}^{x_1}_{y_1}\,][\,{}^{x_2}_{y_2}\,]\cdots[\,{}^{x_n}_{y_n}\,]$ 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 the definition of language recognition, we need to impose certain reasonable *accepting criteria* as well as *rejecting criteria* onto our 1TMs to define the set of "accepted" input strings. With such criteria, we say that a 1TM *recognizes* a language A if, for every string x, (i) if $x \in A$ then M halts on input x and satisfies the accepting criteria and (ii) if $x \notin A$ then M halts and satisfies the rejecting criteria.

The non-regularity measure has played a key role in automata theory. 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 a set in which any distinct pair is n-dissimilar with respect to L [13]. It is immediate from the Myhill-Nerode theorem [20] that a language L is regular if and only if $N_L(n) = O(1)$ [13]. This is further improved by the results of Karp [23] and of Kaņeps and Freivalds [22] 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., [19, 20]). 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) nondeterministic 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*) [38, 40] is a quintuple $N = (Q, \Sigma, \pi, \{T(\sigma) \mid \sigma \in \Sigma\}, \eta)$, where (i) Q is a finite set of states, (ii) Σ is a finite alphabet, (iii) π is a row vector of length $|Q|$ having rational components, (iv) for each $\sigma \in \Sigma$, $T(\sigma)$ is an $|Q| \times |Q|$ matrix whose elements are rational numbers, and (v) η is a column vector of $|Q|$ 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 $|Q|$, 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 $\pi\, T(x)\, \eta$. 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*) [35] N is a rational 1GPFA $(Q, \Sigma, \pi, \{T(\sigma) \mid \sigma \in \Sigma\}, \eta)$ such that (i) π is a stochastic row vector whose entries are all nonnegative, (ii) for each symbol $\sigma \in \Sigma$, $T(\sigma)$ 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 $F = \{a \in Q \mid \text{the } a\text{th entry of } \eta \text{ is } 1\}$. Moreover, since $p_N(x)$ equals the probability of N accepting x, $p_N(x)$ is called the *acceptance probability* of N on the 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} denote the collections of all sets $L(N, \varepsilon)$ for certain rational 1GPFAs N and for certain rational 1PFAs, respectively, where ε is a certain rational number. Similarly, $\mathrm{GSL}^{=}_{rat}$ and $\mathrm{SL}^{=}_{rat}$ are defined from GSL_{rat} and SL_{rat}, respectively, by substituting $L^{=}(N, \varepsilon)$ for $L(N, \varepsilon)$. Sets in SL_{rat} are known as *stochastic* languages [35]. Turakainen [40] 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 one of the most intuitive types of computations. We begin this section with reviewing the major results of Hennie [18] and Kobayashi [25] 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 rather use the following new notations to emphasize our model of one-tape Turing machines. 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 [18] proved that REG = 1-DLIN by employing a so-called *crossing sequence argument*. Elaborating Hennie's argument, Kobayashi [25] substantially improved Hennie's result by showing REG = 1-DTime($o(n \log n)$). This time-bound $o(n \log n)$ is optimal because 1-DTime($O(n \log n)$) contains certain non-regular languages, e.g., $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{a^{2^n} \mid n \in \mathbb{N}\}$. These facts establish the fundamental collapse and separation results concerning deterministic 1TMs.

Proposition 3.1 [18, 25] REG = 1-DTime($o(n \log n)$) $\subsetneqq$ 1-DTime($O(n \log n)$).

In the early 1970s, Bennett [4] 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 adopt the following definition of a (deterministic) reversible Turing machine given by Bernstein and Vazirani [5]. 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 define 1-revDTime($\mathcal{T}$) to be $\bigcup_{T \in \mathcal{T}}$ 1-revDTime($T(n)$). Finally, let 1-revDLIN = 1-revDTime($O(n)$). Obviously, 1-revDLIN is a subset of 1-DLIN.

Kondacs and Watrous [26] 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 besides inputs. Karp and Lipton [24] 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 [8] later considered finite automata that take the Karp-Lipton type advice. To make most of the power of advice, we should take a slightly different formulation for our 1TMs. In this paper, for any complexity class $\mathcal{C}$ defined in terms of Turing machines (including finite state automata as special cases), the notation REG/n is used to represent the collection of all languages A for which there exist an alphabet Σ, a deterministic finite automaton M working with another alphabet, and a total function[§] h from $\mathbb{N}$ to Σ^* with $|h(n)| = n$ (called an *advice function*) satisfying that, for every $x \in \Sigma^*$, $x \in A$ if and only if $[\,{}^{x}_{h(|x|)}\,] \in L(M)$. For instance, the context-free language $L_{eq} = \{0^n 1^n \mid n \in \mathbb{N}\}$ belongs to REG/n. More generally, every language L, over alphabet Σ, whose restriction $L \cap \Sigma^n$ for each length n has cardinality bounded from above by a certain constant, independent of n, belongs to REG/n, because the advice can encode a finite look-up table for length n.

This gives the obvious separation REG $\subsetneqq$ REG/n. On the contrary, REG/n cannot include CFL since, as we see below, the non-regular language $Equal = \{x \in \{0,1\}^* \mid \#_0(x) = \#_1(x)\}$, where $\#_i(x)$ denotes the number of occurrences of the symbol i in x, is situated outside of REG/n. This result will be used in Section 7.

Lemma 3.3 *The language Equal is not in* REG/n. *Hence,* CFL $\nsubseteq$ REG/n.

Proof. Let $\Sigma = \{0,1\}$. Assuming that $Equal \in$ 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 string $x \in \Sigma^*$, $x \in Equal$ if and only if $[\,{}^{x}_{h(|x|)}\,] \in L(M)$. Take $n = |Q|$. For each number $k \in [0,n]_{\mathbb{Z}}$, y_k denotes any string of length n satisfying $\#_0(y_k) = k$.

There exist two *distinct* indices $k, l \in [0,n]_{\mathbb{Z}}$ such that (i) $y_k z_k, y_l z_l \in Equal$ for certain strings $z_k, z_l \in \Sigma^n$ and (ii) M enters the same internal state after reading $[\,{}^{y_k}_{w_n}\,]$ as well as $[\,{}^{y_l}_{w_n}\,]$, where w_n is the first n bits of $h(2n)$. Notice that such a pair (k,l) indeed exists because $n+1 > |Q|$. It follows from these conditions that M also accepts the input $[\,{}^{y_k z_l}_{h(|y_k z_l|)}\,]$. Thus, $\#_0(y_k z_l) = \#_1(y_k z_l)$, which implies $\#_0(z_l) = n - k$. However, since $\#_0(y_l z_l) = \#_1(y_l z_l)$, we obtain $\#_0(y_l) = k$. This contradicts the definition of y_l. Therefore, $Equal$ is not in REG/n. The second claim CFL $\nsubseteq$ REG/n follows from the fact that $Equal \in$ CFL. □

Up to now, we have viewed "Turing machines" as language recognizers (or language acceptors); however, unlike deterministic finite state automata, Turing machines are fully capable of computing partial functions. Since a 1TM M has only one input/work tape, we need to designate the same input tape as the output tape

[§] As standard in computational complexity theory, we allow non-recursive advice functions in general.

of the machine as well. To specify an "outcome" of the machine, we adopt the following convention. When the machine eventually halts with its output tape consisting only of a single block of non-blank symbols, say s, surrounded by the blank symbols, in a way that the leftmost symbol of s is written in the start cell, we consider s 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., [20] for a historical account). In comparison with 1-FLIN, 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 $Q \times \Sigma$ to Γ such that, on input $x = x_1 x_2 \cdots x_n$, 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, where a (total) function is called *length-preserving* if $|f(x)| = |x|$ for any string x. Consider the length-preserving function f defined by $f(x_1 x_2 \cdots x_n) = x_n x_1 \cdots x_{n-1}$ for any $x_1, x_2, \ldots, x_n \in \{0, 1\}$. It is clear that no Mealy machine can compute f. We therefore obtain the following proposition.

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

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($o(n \log n)$) $\subsetneqq$ 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 has Kobayashi's argument in [25, Theorem 3.3] as its core, and the second lemma is due to Hennie [18, Theorem 2]. For the description of the lemmas, we need 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 internal 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, let us assume that the head is scanning tape symbol σ at tape cell i in state p. An application of a transition $(q, \tau, R) \in \delta(p, \sigma)$ makes the machine write symbol τ into cell i, enter state q, and then move the head to cell $i + 1$. The state in which the machine crosses the intercell boundary between cell i and cell $i + 1$ is q (not p). Similarly, if we apply a

[¶] As stated in Section 2, this paper accommodates the *strong definition* of running time; namely, the running time of a machine M on input x is the height of the computation tree produced by M on x, independent of the outcome of the computation.

transition $(q, \tau, L) \in \delta(p, \sigma)$, then q is the state in which the machine crosses the intercell 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 observes that Kobayashi's argument extends to nondeterministic 1TMs without depending on their acceptance criteria. 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 any critical boundary in any (accepting or rejecting) computation path of M on the input x has length at most c.*

In essence, Hennie [18] 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($o(n \log n)$) by Theorem 4.1. In contrast, a simple crossing-sequence argument proves that 1-NTime($O(n \log n)$) does not contain the set of all palindromes, $Pal = \{x \in \{0,1\}^* \mid x = x^R\}$, where x^R is the *reverse* of x. Since $\overline{Pal} \in$ 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 between two languages has played a central role in the theory of NP-completeness as a measuring tool for the complexity of languages. We can see reducibility as a basis of "relativization" with oracles. For instance, Turing reducibility 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 language A over alphabet Σ_1 is said to be *many-one 1-NLIN-reducible to* another language B over alphabet Σ_2 (notationally, $A \leq_m^{\text{1-NLIN}} B$) if there exist a linear function T and a nondeterministic 1TM M such that, for every string x in Σ_1^*, (i) M on the input x halts within time $T(|x|)$ with the tape consisting only of one block of non-blank symbols, say y_p, on every computation path p, provided that the left-most symbol of y_p must be written in the start cell, (ii) when M eventually halts, the tape head returns to the start cell along all computation paths, and (iii) $x \in A$ if and only if $y_p \in B$ for some *accepting* computation path p of M on the input x. For any fixed set B, 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, for any complexity class $\mathcal{C}$, the notation 1-NLIN$_m^{\mathcal{C}}$ stands for the union of sets 1-NLIN$_m^B$ over all sets B in $\mathcal{C}$.

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. This proposition is essentially equivalent to the transitive property of the relation $\leq_m^{\text{1-NLIN}}$. Let A, B, and C be three arbitrary languages and assume that $A \leq_m^{\text{1-NLIN}} B \leq_m^{\text{1-NLIN}} C$. Our goal is 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 if and when it halts with an admissible value on the tape, start M' on that value as its input. This machine N is clearly nondeterministic and its running time is $O(n)$ since so are the running times of M and M'. It is not difficult to check 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$_m^B$ for any set B. Although 1-DLIN = 1-NLIN, two reducibilities, many-one 1-NLIN-reducibility and many-one 1-DLIN-reducibility, are quite different in their power. As an example, we can construct a recursive set B that separates between 1-DLIN$_m^B$ and 1-NLIN$_m^B$. The construction of such a set B can be done by a standard diagonalization technique.

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

Proof. For any set $B \subseteq \{0,1\}^*$, 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 set B.

Baker, Gill, and Solovay [2] constructed a recursive set B such that L_B cannot be polynomial-time Turing reducible to B. In particular, L_B is not many-one 1-DLIN-reducible to B; that is, $L_B \notin$ 1-DLIN$_m^B$. □

4.2 Multi-Valued Partial Functions

Conventionally, a Turing machine that can output values is called a *transducer*. Nondeterministic transducers can compute multi-valued partial functions in general. Let us consider a nondeterministic 1TM that outputs a certain string in Σ_2^* (whose leftmost symbol is in the start cell) along each computation path by entering a certain halting state. Similar to partial functions introduced in Section 3, we invalidate any rejecting computation path and let $M(x)$ denote the set of all *valid* outcomes of M on input x. In particular, when M on the input x enters the rejecting state along all computation paths, $M(x)$ becomes the empty set. 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 string $x \in \Sigma_1^*$. Let 1-NLINSV be the subset of 1-NLINMV, containing only single-valued partial functions. In contrast, 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$_t$.

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

The basic relationship between functions in 1-FLIN and languages in 1-DLIN is stated in Lemma 4.7. 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. Moreover, for any multi-valued partial function $f \in$ LPF, let $L[f] = \{[\,{}^x_y\,] \mid y \in f(x)\}$.

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 $[\,{}^x_y\,]$, 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 $[\,{}^x_y\,]$. If N accepts, output y. Clearly, M computes f and thus f is in 1-NLINMV. □

The following major collapse result 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 $g(x) = \emptyset$. (See, e.g., Selman's paper [37] for this notion.)

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 contents of cells in its input area, where the *input area* means the tape region where given input symbols are initially written. For later use, we give a general description of a folding machine.

Construction of a Folding Machine. Let $M = (Q, \Sigma, \Gamma, \delta, q_0', q_{acc}, q_{rej})$ be any 1TM that always halts in linear time. The *folding machine* N is constructed from M as follows. Choose the minimal positive integer k

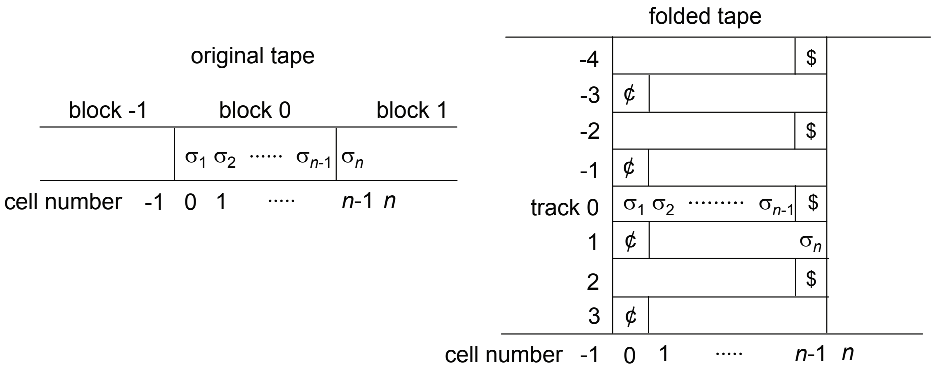

Figure 1: The tape design of the folding machine N where $k = 2$. The original tape of M is partitioned into $4k$ blocks of size $n - 1$ and each block is simulated by a track in the folded tape of N. For instance, block 0 is simulated by track 0, and block 1 is simulated by track 1 in the reverse order.

such that $\mathrm{Time}_M(x) \leq k|x|$ for all inputs x of length at least 3. Notice that, since M's tape head moves in both directions on its tape, M can use tape cells indexed between $-2k(|x| - 1)$ and $2k(|x| - 1) - 1$. Choose four new internal states q_0, q_1, q_2, q_3 not in Q and introduce new internal states of the form $[\,\begin{smallmatrix} i \\ q \end{smallmatrix}\,]$ for each number $i \in [-2k, 2k - 1]_{\mathbb{Z}}$ and each internal state $q \in Q$. Let x be an arbitrary string written in the input tape of M.

1) The machine N starts in the new 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 symbol in Σ. Note that σ_1 is written in the start cell.

2) In this preprocessing phase, the machine N re-designs its input/work tape, as shown in Figure 1, by moving its head. In the original tape of M, the cells indexed between $-2k(|x| - 1)$ and $2k(|x| - 1) - 1$ are partitioned into $4k$ blocks of $|x| - 1$ cells. These blocks are indexed in order from the leftmost block to the rightmost block using integers ranging from $-2k$ to $2k-1$. In particular, block 0 contains the string $\sigma_1\sigma_2 \cdots \sigma_{n-1}$ (without σ_n). We split the tape of N into $4k$ tracks, which are indexed from the top to the bottom using $-2k$ to $2k - 1$. Intuitively, we want to simulate block i of M's tape using track i of N's folded tape. The machine N first places the special symbol ¢ (*left end-marker*) in all tracks of odd indices and then enters the internal state q_1 by stepping right. The machine keeps moving its head rightward in the state q_1. When the head encounters the first blank symbol, if $|x| \geq 3$ then N enters the state q_2 and steps back; otherwise, N enters M's halting state. In a single step, N places another special symbol \$ (*right end-marker*) in all tracks of even indices, shifts σ_n in track 0 to track 1, enters the state q_3, and steps to the left. The head then returns to the start cell in state q_3. Notice that this phase can be 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 $[\,\begin{smallmatrix} i \\ q \end{smallmatrix}\,]$. 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 the state q, N instead enters state $[\,\begin{smallmatrix} 2j+1 \\ q \end{smallmatrix}\,]$ ($[\,\begin{smallmatrix} 2j-1 \\ q \end{smallmatrix}\,]$, 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. It is clear that N's head never visits outside of the input area. This simulation phase takes exactly the same amount of time as M's.

Consider the set S of all (possible) crossing sequences of the folding machine N. For any two crossing sequences $v, v' \in S$ and any tape symbol σ, we write $v \rightarrow_\sigma v'$ 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. Along 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. 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 $v_{i-1} \rightarrow_{\sigma_i} v_i$ for every index $i \in [1, n]_{\mathbb{Z}}$.

Now, let us 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 $M^* = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ that computes f. Consider the folding machine N constructed from M^*. Matching the output convention of 1TMs, we need to modify this folding machine to produce the outcomes of M^*. After N eventually halts, we further move the tape head leftward. When we reach the left end-marker, we move back the head by changing the current tape symbol to the symbol written in the area of track 0 and track 1 where the original input symbols of N is written. When we reach the right end-marker, we step right to the first blank symbol by entering a halting state (either q_{acc} or q_{rej}) of M^*. Evidently, this modified nondeterministic 1TM produces the outcomes of M^* and also enters exactly the same halting states of M^*. This modified machine is hereafter referred to as N for our convenience.

Let CS be the set of all crossing sequences of N. Assume that all elements in CS are enumerated so that we can always find the minimal element in any subset of CS. For any two elements $v, v' \in CS$ and any symbol σ with $v \to_\sigma v'$, $Symb(v, \sigma, v')$ denotes the output symbol written in the cell where σ is initially written. This symbol $Symb(v, \sigma, v')$ can be easily deduced from (v, v', σ) by tracing the tape head moves crossing a cell that initially contains the symbol σ.

Finally, we want to construct a refinement g of f. This desired partial function g is defined by a deterministic 1TM M that behaves as follows. Let $n \in \mathbb{N}$ and let $x = \sigma_1\sigma_2\cdots\sigma_n$ be an arbitrary input of length n. Set $v_0 = ()$ and $v_f = (q_1, q_2)$ as before.

1) In this phase, all internal states except v_f are subsets of CS. Let $S_1 = \{v_0\}$ be the initial state of M. Let $i \in [1, n]_{\mathbb{Z}}$ and assume that M currently scans the input symbol σ_i in internal state S_i. We define two key sets $V_i = \{v \in S_i \mid \exists\, v' \in CS\ [v \to_{\sigma_i} v']\}$ and $S_{i+1} = \{v' \in CS \mid \exists\, v \in V_i\ [v \to_{\sigma_i} v']\}$. Intuitively, S_{i+1} captures all possible nondeterministic moves from S_i. Notice that $V_i \subseteq S_i$. When S_{i+1} is empty, M enters a new rejecting state. Provided that S_{i+1} is non-empty, M changes the tape symbol σ_i to $[\begin{smallmatrix}\sigma_i\\ V_i\end{smallmatrix}]$ and enters S_{i+1} as an internal state by stepping to the right. Unless $x \notin \mathrm{dom}(f)$, after scanning σ_n, M enters the internal state S_{n+1}. By the property of the original folding machine, we must have $S_{n+1} = \{v_f\}$. For later convenience, let $v_n = v_f$ and $V_{n+1} = S_{n+1}$. When the tape head scans the first blank symbol, M then enters the internal state v_n by stepping to the left.

2) In the beginning of this second phase, M is in the state v_n, scanning the rightmost tape symbol $[\begin{smallmatrix}\sigma_n\\ V_n\end{smallmatrix}]$ in the input area. Notice that $v_n \in V_{n+1}$. For any index $i \in [1, n]_{\mathbb{Z}}$, let us assume that M scans the symbol $[\begin{smallmatrix}\sigma_i\\ V_i\end{smallmatrix}]$ in the state v_i, where $v_i \in V_{i+1} \subseteq S_{i+1}$. Since M passes the first phase and enters the second phase, V_i cannot be empty. Since $v_i \in S_{i+1}$, the set $W_i = \{v \in V_i \mid v \to_{\sigma_i} v_i\}$ is not empty, either. Choose the minimal element, say v_{i-1}, in W_i. This crossing sequence v_{i-1} obviously satisfies that $v_{i-1} \to_{\sigma_i} v_i$. Now, M changes the symbol $[\begin{smallmatrix}\sigma_i\\ V_i\end{smallmatrix}]$ to $Symb(v_{i-1}, \sigma_i, v_i)$ and moves its tape head to the left by entering v_{i-1} as an internal state. After scanning $[\begin{smallmatrix}\sigma_1\\ V_1\end{smallmatrix}]$, M enters the internal state v_0 because $V_1 = S_1 = \{v_0\}$. Note that the resulting series $(v_0, v_1, v_2, \ldots, v_n)$ specifies a certain accepting computation path of N and the output tape of M contains the outcome produced along this particular computation path. When the tape head reaches the blank symbol, M finally enters a new accepting state. This completes the description of M.

The above deterministic 1TM M clearly produces, for each input x, at most one output string from the set $f(x)$. Note that, if $x \notin \mathrm{dom}(f)$, all computation paths are rejecting paths, and thus M never reaches any accepting state. It is therefore obvious that the partial function g computed by M is a refinement of f. □

Another application of Lemma 4.9 is the non-existence of one-way functions in 1-FLIN. To describe the notion of one-way function in our single-tape linear-time model, we need to expand our "track" notation $[\begin{smallmatrix}x\\ y\end{smallmatrix}]$ to the case where $|x|$ and $|y|$ differ. To keep our notation simple, we also use the same notation $[\begin{smallmatrix}x\\ y\end{smallmatrix}]$ to express $[\begin{smallmatrix}x\#^k\\ y\end{smallmatrix}]$ if $|x| + k = |y|$ and $k \geq 1$ and express $[\begin{smallmatrix}x\\ y\#^k\end{smallmatrix}]$ if $|x| = |y| + k$ and $k \geq 1$, where $\#$ is a distinct "blank" symbol. 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 $f\left(g\left([\begin{smallmatrix}f(x)\\ 1^{|x|}\end{smallmatrix}]\right)\right) = f(x)$ for all inputs x. When f is length-preserving, the equality $f\left(g\left([\begin{smallmatrix}f(x)\\ 1^{|x|}\end{smallmatrix}]\right)\right) = f(x)$ can be replaced by $f(g(f(x))) = f(x)$.

Proposition 4.10 *There is no one-way function in* 1-FLIN.

Proof. Assume that a one-way function f mapping Σ_1^* to Σ_2^* exists in 1-FLIN. Let f^{-1} denote a multi-valued partial function defined as follows. For each string of the form $[\begin{smallmatrix}y\\ 1^n\end{smallmatrix}]$, if $|y| \geq n$, then we define $f^{-1}\left([\begin{smallmatrix}y\\ 1^n\end{smallmatrix}]\right) = \{x\#^{|y|-n} \mid |x| = n, f(x) = y\}$; otherwise, let $f^{-1}\left([\begin{smallmatrix}y\\ 1^n\end{smallmatrix}]\right) = \{x \mid |x| = n, f(x) = y\}$. 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 $[\begin{smallmatrix}y\\ 1^n\end{smallmatrix}]$, check if $[\begin{smallmatrix}y\\ 1^n\end{smallmatrix}] \in \mathrm{dom}(g)$. If not, M outputs any fixed string of length n (e.g., 0^n). Otherwise, M computes $g\left([\begin{smallmatrix}y\\ 1^n\end{smallmatrix}]\right)$ and outputs a string obtained from it by deleting the symbol $\#$. Since $\mathrm{dom}(g)$ is in 1-DLIN, M can be deterministic. Clearly, M

inverts f; that is, $f\left(M\left(\left[\begin{smallmatrix} f(x) \\ 1^{|x|}\end{smallmatrix}\right]\right)\right) = f(x)$ for all inputs x. This contradicts the one-wayness of f. Therefore, f cannot be one-way. □

The third application concerns the advised class REG$/n$. Similar to this class, we define 1-DLIN/lin as the collection of all languages A such that there are a linear-time deterministic 1TM M, an advice function h, and a constant $c \geq 1$ for which (i) $|h(n)| \leq cn + c$ for any number $n \in \mathbb{N}$, and (ii) for every x, $x \in A$ iff $\left[\begin{smallmatrix} x \\ h(|x|)\end{smallmatrix}\right] \in L(M)$. Now, we can prove that the two classes REG$/n$ and 1-DLIN/lin coincide.

Proposition 4.11 REG$/n$ = 1-DLIN/lin.

Proof. The inclusion REG$/n \subseteq$ 1-DLIN/lin is obvious. Now, we want to show that 1-DLIN/*lin* $\subseteq$ REG$/n$. Let A be any language, over alphabet Σ, in 1-DLIN/lin. Without loss of generality, we can take a linear-time deterministic 1TM M and an advice function h satisfying that (i) $n \leq |h(n)| \leq cn$ for any number $n \in \mathbb{N}$ and (ii) for every x, $x \in A$ iff $\left[\begin{smallmatrix} x \\ h(|x|)\end{smallmatrix}\right] \in L(M)$. For simplicity, we assume that an alphabet for our advice strings is different from Σ.

Let x be any input of length n. Initially, the tape of M consists of the string $\left[\begin{smallmatrix} x\#^{|h(n)|-n} \\ h(n)\end{smallmatrix}\right]$. A folding machine N, induced from M, starts with its own input, say $cont(x, h(n))$, which is obtained by folding the tape content $\left[\begin{smallmatrix} x\#^{|h(n)|-n} \\ h(n)\end{smallmatrix}\right]$. From this input string $cont(x, h(n))$, we can construct another string simply by deleting all symbols in Σ. Since this new string does not include x, we denote it by $h'(n)$. Note that $|h'(n)| = n$.

Let us describe a new deterministic 1TM M' that behaves as follows. On input $\left[\begin{smallmatrix} x \\ h'(|x|)\end{smallmatrix}\right]$, M' first modifies the input to $cont(x, h(|x|))$ in linear time and then simulates the folding machine N using this new string as an input. Obviously, for every string x, $x \in A$ iff M' accepts $\left[\begin{smallmatrix} x \\ h'(|x|)\end{smallmatrix}\right]$. Since M' runs in linear time using only its input area, we can translate M' into its equivalent deterministic finite automaton. Therefore, we can conclude that A belongs to REG$/n$. □

5 Alternating Computation

Chandra, Kozen, and Stockmeyer [6] 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 using our strong definition of running time. An *alternating 1TM* is defined similar to a nondeterministic 1TM except that its internal states are all labeled with symbols in $\{\exists, \forall\}$, where $\exists$ reads "existential" and $\forall$ reads "universal" (this labeling is done by a fixed function that maps the set of internal states to $\{\exists, \forall\}$). 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 in the following recursive fashion. A leaf is evaluated T if and only if it is in the accepting state. An internal node labeled with symbol $\exists$ is evaluated T if and only if at least one of its children is evaluated T. An internal node labeled with symbol $\forall$ is evaluated T if and only if all of its children are evaluated T. 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 every computation path. For instance, a nondeterministic Turing machine can be viewed as an alternating Turing machine whose internal states are all labeled $\exists$, and therefore it has 1-alternation. Let k and T be any functions from $\mathbb{N}$ to $\mathbb{N}$ satisfying that $k(n) \leq T(n)$ for all $n \in \mathbb{N}$. The notation 1-$\Sigma_{k(n)}$Time$(T(n))$ (1-$\Pi_{k(n)}$Time$(T(n))$, resp.) expresses the collection of all languages recognized by certain $T(n)$-time alternating 1TMs with *at most* $k(n)$-alternation starting with an $\exists$-state (a $\forall$-state, resp.). For any given language $A \in$ 1-$\Sigma_{k(n)}$Time$(T(n))$, take a $T(n)$-time alternating 1TM M that recognizes A with at most $k(n)$-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. It follows that $\overline{M}$ is a $T(n)$-time alternating 1TM with at most $k(n)$-alternation starting with a $\forall$-state. Clearly, $\overline{M}$ recognizes $\overline{A}$. Thus, co-1-$\Sigma_{k(n)}$Time$(T(n)) \subseteq$ 1-$\Pi_{k(n)}$Time$(T(n))$. Similarly, we have co-1-$\Pi_{k(n)}$Time$(T(n)) \subseteq$ 1-$\Sigma_{k(n)}$Time$(T(n))$, and hence 1-$\Pi_{k(n)}$Time$(T(n))$ = co-1-$\Sigma_{k(n)}$Time$(T(n))$. Given a set $\mathcal{T}$ of time-bounding functions, 1-$\Sigma_{k(n)}$Time$(\mathcal{T})$ (1-$\Pi_{k(n)}$Time$(\mathcal{T})$, resp.) stands for the union of all sets 1-$\Sigma_{k(n)}$Time$(T(n))$ (1-$\Pi_{k(n)}$Time$(T(n))$, resp.) over all functions T in $\mathcal{T}$. In particular, we write 1-$\Sigma^{\mathrm{LIN}}_{k(n)}$ (1-$\Pi^{\mathrm{LIN}}_{k(n)}$, resp.) for 1-$\Sigma_{k(n)}$Time$(O(n))$ (1-$\Pi_{k(n)}$Time$(O(n))$, resp.).

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 1-Π^{LIN}_{k} = co-1-$\Sigma^{\mathrm{LIN}}_{k}$ and REG $\subseteq$ 1-$\Sigma^{\mathrm{LIN}}_{k} \cap$ 1-$\Pi^{\mathrm{LIN}}_{k} \subseteq$ 1-$\Sigma^{\mathrm{LIN}}_{k} \cup$ 1-$\Pi^{\mathrm{LIN}}_{k} \subseteq$ 1-ALIN.

Similarly, we define the complexity class 1-Δ_k^{LIN} by 1-$\Delta_{k+1}^{\mathrm{LIN}}$ = 1-$\mathrm{DLIN}_m^{1\text{-}\Sigma_k^{\mathrm{LIN}}}$ for every index $k \in \mathbb{N}$. Now, we generalize the earlier collapse result 1-NLIN = REG and prove that three complexity classes 1-Σ_k^{LIN}, 1-Π_k^{LIN}, and 1-Δ_k^{LIN} all collapse to REG.

Theorem 5.1 REG $= \bigcup_{k\in\mathbb{N}^+}$ 1-Σ_k^{LIN} $= \bigcup_{k\in\mathbb{N}^+}$ 1-Π_k^{LIN} $= \bigcup_{k\in\mathbb{N}^+}$ 1-Δ_k^{LIN}.

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 *For every number $k \in \mathbb{N}^+$, 1-$\Sigma_{k+1}^{\mathrm{LIN}}$ = 1-$\mathrm{NLIN}_m^{1\text{-}\Pi_k^{\mathrm{LIN}}}$.*

The proof of Theorem 5.1 proceeds by induction on k. The base case $k = 1$, i.e., 1-Π_1^{LIN} = 1-Σ_1^{LIN} = REG, 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-Σ_k^{LIN}. Lemma 5.2 yields the existence of a set $B \in$ 1-Π_{k-1}^{LIN} satisfying that $A \in$ 1-NLIN_m^B. By the induction hypothesis, B falls into REG and hence A is in 1-$\mathrm{NLIN}_m^{\mathrm{REG}}$, which is obviously REG. Since 1-Π_k^{LIN} = co-1-Σ_k^{LIN}, 1-Π_k^{LIN} also collapses to REG. Similarly, from the inclusion 1-$\Delta_k^{\mathrm{LIN}} \subseteq$ 1-Σ_k^{LIN}, it follows that 1-Δ_k^{LIN} = REG.

Proof of Lemma 5.2. ($\subseteq$-direction) Let A be any language in 1-$\Sigma_{k+1}^{\mathrm{LIN}}$, where $k \geq 1$, over alphabet Σ_1. Take a linear-time alternating 1TM M with at most k-alternation that recognizes A. 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 a linear-time nondeterministic 1TM M' that simulates M during its first alternation. Let # be a new symbol not in Σ_1. On input x, M' marks the start cell and then starts simulating M. During this simulation, whenever M writes a blank symbol, M' replaces it with #. When M enters the first $\forall$-state p, M' first marks the currently scanning cell (by changing its tape symbol a to the new symbol $[\,{}^{a}_{p}\,]$), moves its tape head back to the start cell, and finally erases the mark at the start cell. It is important to note that M' has only one block of at least $|x|$ non-blank symbols on its tape when it halts. Let Σ consist of all symbols of the form $[\,{}^{a}_{p}\,]$, where a is any symbol in Σ_1 and p is any $\forall$-state.

Next, we define another alternating 1TM N as follows. Let $\Sigma_2 = \Sigma_1 \cup \Sigma \cup \{\#\}$ be a new alphabet for N. On input y in Σ_2^*, N changes all #s to the blank symbol, finds on the tape the leftmost cell that contains a symbol, say $[\,{}^{a}_{p}\,]$, from Σ, and changes it back to a. At the same time, M' recovers the $\forall$-state p as well. By starting with this internal state p of M, N simulates M step by step. Finally, the desired set B is defined to include all input strings accepted by N. Obviously, B is in 1-Π_{k-1}^{LIN}. By their definitions, M' many-one 1-NLIN-reduces A to B.

($\supseteq$-direction) Assume that A is in 1-$\mathrm{NLIN}_m^{1\text{-}\Pi_k^{\mathrm{LIN}}}$; namely, A is many-one 1-NLIN-reducible to B via a reduction machine N, where B is a certain set in 1-Π_k^{LIN}. Choose a linear-time alternating 1TM M that recognizes B with k-alternation starting with a $\forall$-state. Now, let us define N' as follows: on input x, simulate N, and, when N eventually halts, simulate M on the same input/work tape. Clearly, N' runs in linear time since so do M and N. It is also easy to show that N' recognizes A with $(k+1)$-alternation starting with an $\exists$-state. Thus, A belongs to 1-$\Sigma_{k+1}^{\mathrm{LIN}}$. □

The collapse of the hierarchy of alternating complexity classes with constant-alternation depends on our strong definition of nondeterministic running time. By contrast, when the linear-time alternating class is defined with a *weak definition* of running time (e.g., the length of the shortest accepting path if one exists, and 1 otherwise), the language $L = \{x\#y \mid x, y \in \Sigma^*,\ y \text{ is the binary representation of } |x|\}$ can separate this alternating class from REG. (See [33] also [3].)

6 Probabilistic Computation

Probabilistic (or randomized) computation has been proven to be essential to many applications in computer science. Since as early as the 1950s, probabilistic extensions of deterministic Turing machines have been studied from theoretical interest as well as for practical applications. This paper adopts Gill's model of probabilistic Turing machines with flipping *fair coins* [16]. Formally, we define a *probabilistic 1TM* as a nondeterministic 1TM that has *at most* two nondeterministic choices at each step, which is referred to as a *coin toss* (or *coin flip*) whenever there are exactly two choices. Each fair coin toss is made with probability exactly 1/2. Instead of taking an expected running time, we define a probabilistic 1TM M to be *$T(n)$-time bounded* if, for each string x, all computation paths of M on the input x have length at most $T(|x|)$. This definition reflects our strong definition of running time. The probability associated with each computation path s equals 2^{-m}, where m is the number of coin tosses along the path s. The *acceptance probability* of M on the input x, denoted $p_M(x)$, is the sum of the probabilities of all accepting computation paths. For any language L, we say that M *recognizes L with error probability at most ε* if, for every x, (i) if $x \in L$, then $p_M(x) \geq 1 - \varepsilon$; 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. Kaņeps and Freivalds [22], following Rabin's [35] 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 number $n \in \mathbb{N}$, let S_n be the union, over all strings x of length at most n, of the sets of all crossing sequences at any critical-boundary of x along any accepting computation path of M on x. 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 $w_l(x|v)$ be the sum, over all z with $|xz| \leq n$, of all probabilities of the coin tosses made during the tape head staying in the left-side region of the right-boundary of x along any accepting computation path of M on the input xz. Similarly, for every $z \in \Sigma^{\leq n}$ and every $v \in S_n$, let $w_r(v|z)$ be the sum, over all x with $|xz| \leq n$, of all probabilities of the coin tosses made during the tape head staying in the right-side region of the left-boundary of z along any accepting computation path of M on the input xz. By these two definitions, it follows that $0 \leq w_l(x|v), w_r(v|z) \leq 1$. The key observation is that the acceptance probability of M on the 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 support set $\mathrm{Supp}_n(x) = \{(i, v) \mid x \text{ } n\text{-supports } (i, v)\}$. 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 crossing sequences $v \in S_n$. 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 $p_M(yz) > \varepsilon(n)$. Hence, we obtain $yz \in L$.

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

Let us focus our attention on the case where the error probability of a probabilistic 1TM is bounded away from 1/2. For each 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 $1/2 - \varepsilon$. 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($\mathcal{T}$) for any set $\mathcal{T}$ of time-bounding functions. The *one-tape bounded-error probabilistic linear-time* class 1-BPLIN is 1-BPTime($O(n)$).

Consider any language L recognized by a probabilistic 1TM M with bounded-error probability in time $o(n \log n)$. Lemma 4.2 implies that the number of all crossing sequences of M is upper-bounded by a certain constant independent of its input. It thus follows from Lemma 6.1 that $N_L(n)$ is bounded above by an exponential function of the machine's error bound $\varepsilon(n)$. Since $\varepsilon(n)$ is bounded away from 1/2, we obtain $N_L(n) = O(1)$, which yields the regularity of L. Therefore, REG = 1-BPTime($o(n \log n)$). The separation REG $\neq$ 1-BPTime($O(n \log n)$) follows from Proposition 3.1.

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

Leaving from bounded-error probabilistic computation, we hereafter concentrate 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 a 1TM-characterization of SL_{rat}. A similar characterization of SL_{rat} was given by Kaņeps [21] in terms of one-head two-way probabilistic automata with rational transition probabilities. For simplicity, 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}.

Now, let us prove Theorem 6.3. The theorem follows from two key lemmas: Lemmas 6.4 and 6.5. We begin with Lemma 6.4 whose proof is based on a simple simulation of rational 1PFAs by synchronous probabilistic 1TMs.

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

Proof. Let L be any language in SL_{rat}. There exists a rational 1PFA $N=(S,\Sigma,\pi,\{T(\sigma)\,|\,\sigma\in\Sigma\},\eta)$ with a rational cut point for L. 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) one entry of π equals 1, and (iv) there is a positive integer d satisfying the following property: for any symbol $\sigma\in\Sigma$ and any pair $i,j\in S$, the (i,j)-entry of the matrix $T(\sigma)$, denoted $T(\sigma)_{i,j}$, is of the form $r_{i,j}(\sigma)/d$ for a certain number $r_{i,j}(\sigma)\in\mathbb{N}$. Write F for the set of all the final states of N.

Our goal is to construct a synchronous probabilistic 1TM M that simulates N in linear time with unbounded error. The desired machine M works as follows. In case where the input is the empty string λ, 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 λ, we give an algorithmic description of N's behavior. Choose an integer m such that $2^{m-1}<d\leq 2^{m}$. First, we repeat phases 1)-2) until M finishes scanning all input symbols. Initially, M sets its *decision value* to be -1.

1) In scanning a symbol σ, M first generates 2^{m} branches by tossing exactly m fair coins without moving its head. The (lexicographically) first d branches are called *useful*; the other branches are called *useless*. The useful branches are used for the simulation of a single step of N in phase 2.

2) First, we consider the case where the current decision value is -1. In the following manner, M simulates a single step of N's moves. Assume that N is in internal state i. Note that, for any choice $j\in S$, N changes the internal state i to j with the transition probability $T(\sigma)_{i,j}$ $(=r_{i,j}(\sigma)/d)$ while scanning the symbol σ. To simulate such a transition, we choose exactly $r_{i,j}(\sigma)$ branches out of the useful d branches and then follow the same transition of N. More precisely, along the ℓth branch generated by the coin tosses made in phase 1, M simulates N's transition from the internal state i to 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-adjacent cell. Along the useless branches, M tosses a fair coin, remembers its outcome (either 0 or 1) as a new decision value, and moves its head rightward. If the current decision value is not -1, then we simply force M's head to step to the right.

3) When M finishes reading the entire input, its head must sit in the first blank cell. With the decision value -1, if N reaches a final state in F, then M enters q_{acc}; otherwise, M enters q_{rej}. If the decision value is either 0 or 1, M enters q_{rej} or q_{acc}, respectively. This completes the description of M.

By our simulation, the acceptance probability of N on input x is greater than $1/2$ iff the acceptance probability of M on the same input is more than $1/2$. Moreover, our simulation makes M's computation paths terminate all at once. 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 $\subseteq\mathrm{SL}_{rat}$, take a language L in 1-PLIN and consider any 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}. With Lemmas 6.4, we therefore obtain Theorem 6.3.

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 as M does. Note that the accepting probability of this modified machine is the same as that of M. For notational simplicity, we use the notation M to denote this modified machine. For this new machine M, the crossing sequence at the left-boundary of any input should be $v_0=()$ and the crossing sequence at the right-boundary of any input is $v_f=(q_1,q_2,q_{acc})$ along every accepting computation path of M.

We wish to construct a rational 1GPFA $N=(S,\Sigma,\pi,\{T(\sigma)\,|\,\sigma\in\Sigma\},\eta)$ satisfying that $p_N(x)=p_M(x)$ 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 tree of M where M starts on input $y\sigma z$ with the tape 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 this computation tree, (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) $u \rightarrow_\sigma v$.

Clearly, $P(u; \sigma; v)$ is a dyadic 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 tape symbol $\sigma \in \Sigma$, define $T(\sigma)$ to be the $|S| \times |S|$ matrix whose (u, v)-element is $P(u; \sigma; v)$ for any pair $u, v \in S$. The row vector π has 1 or 0 in the vth column if $v = v_0$ or $v \neq v_0$, respectively, for any $v \in S$. 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 or 0 if $v \in F$ or $v \notin F$, respectively. Thus, we have $p_N(x) = \pi\, T(x)\, \eta$ for every input string x.

By the above definition of N, it is not difficult to verify that, for each input x, $p_N(x) = \pi\, T(x)\, \eta = \sum_{\vec{v}} \prod_{i=1}^{n} P(v_{i-1}; \sigma_i; v_i) = p_M(x)$, as requested. □

Macarie [28] 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 multiple read/write work-tapes, which uses $O(\log n)$ tape-space on all the tapes except for the input tape (and halting eventually on all inputs). We thus obtain the following consequence of Theorem 6.3. Note that $\mathrm{L} \nsubseteq \mathrm{CFL}$ since, for instance, $L_{3eq} = \{a^n b^n c^n \mid n \in \mathbb{N}\} \in \mathrm{L} - \mathrm{CFL}$.

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

Proof. The proper inclusion 1-PLIN $\subsetneqq$ L follows from Theorem 6.3 as well as the fact that $\mathrm{SL}_{rat} \subsetneqq \mathrm{L}$. Since REG/$n$ contains all non-recursive tally languages, it immediately follows that REG/$n \nsubseteq$ L. To prove that L $\nsubseteq$ REG/n, we use the language $Equal = \{w \in \{0,1\}^* \mid \#_0(w) = \#_1(w)\}$. While Lemma 3.3 places *Equal* outside of REG/n, *Equal* obviously belongs to L. We therefore obtain the last separation L $\nsubseteq$ REG/n. □

Theorem 6.3 also provides us with separations among three complexity classes REG/n, CFL, and 1-PLIN. We see these separation results in the next proposition.

Proposition 6.7 *$\mathrm{CFL} \cap \mathrm{REG}/n \nsubseteq$ 1-PLIN, CFL $\nsubseteq$ 1-PLIN $\cup$ REG/n, and REG/$n \nsubseteq$ CFL $\cup$ 1-PLIN.*

Earlier, Nasu and Honda [31] found a context-free language not in SL_{rat}. More precisely, they introduced the context-free language $L_{NH} = \{a^i b a^{j_1} b \ldots b a^{j_r} b \mid r \in \mathbb{N}^+ \,\&\, i, j_1, \ldots, j_r \in \mathbb{N} \,\&\, i = \sum_{k=1}^{\ell} j_k \text{ for a certain number } \ell \in [1, r]_{\mathbb{Z}}\}$ and showed that, using the Cayley-Hamilton theorem, L_{NH} cannot belong to SL_{rat}. A similar technique can show that SL_{rat} does not contain the context-free language $Center = \{x1y \mid x, y \in \{0,1\}^*, |x| = |y|\}$.

Proof of Proposition 6.7. It is easy to show that the context-free language *Center* falls into REG/n by choosing advice of the form $0^n 1 0^n$ whenever the length $|x1y|$ is odd. Since $Center \notin \mathrm{SL}_{rat}$, the first separation follows from Theorem 6.3.

For the second separation, let us consider the context-free non-stochastic language L_{NH}. It is enough to prove that $L_{NH} \notin \mathrm{REG}/n$. This can be done in a similar fashion as in the proof of Lemma 3.3. Let us fix our alphabet $\Sigma = \{a, b\}$. Assuming that $L_{NH} \in \mathrm{REG}/n$, choose a deterministic finite automaton $M = (Q, \Sigma, q_0, F)$ and an advice function h mapping from $\mathbb{N}$ to Σ^* such that, for every $x \in \Sigma^*$, $x \in L_{NH}$ if and only if $\left[\begin{smallmatrix} x \\ h(|x|) \end{smallmatrix}\right] \in B$. Let $n = |Q| + 2$. For each number $k \in [1, n-1]_{\mathbb{Z}}$, define y_k and z_k to be the strings $a^{n+k} b a^{n-k} b$ and $a^{2k} b a^{2(n-k)} b$ of length $2n+2$, respectively. Obviously, $y_k z_k$ belongs to L_{NH} since $|a^{n+k}| = |a^{n-k}| + |a^{2k}|$. Choose two *distinct* indices $k, l \in [1, n-1]_{\mathbb{Z}}$ such that 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 thus follows that M should accept the input $\left[\begin{smallmatrix} y_k z_l \\ h(|y_k z_l|) \end{smallmatrix}\right]$. This yields the membership $y_k z_l \in L_{NH}$. Clearly, this contradicts the definition of L_{NH}. Therefore, we conclude that $L_{NH} \notin \mathrm{REG}/n$.

The third separation is rather obvious because REG/n contains a non-recursive language. □

7 Counting Computation

Counting issues naturally arise in many fields of computer science. For instance, the decision problem of determining whether there exists a Hamiltonian cycle in a given graph induces the problem of counting the number of such cycles. In the late 1970s, Valiant [43] introduced the notion of *counting Turing machines* to study the complexity of counting. Our goal is to investigate the 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, we take the number of all accepting computation paths as the outcome of the machine. Let $\#M(x)$ denote the outcome of such a counting 1TM M on input x. In this way, counting 1TMs can compute (partial) functions mapping strings to numbers. These functions are called *counting functions*.

Similar to Valiant's function class #P [43], we use the notation 1-#LIN (pronounced "one sharp lin") to denote the collection of all total functions f, from Σ^* to $\mathbb{N}$, which are computed by certain linear-time counting 1TMs. This function 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 off into n computation paths, starting with the nth string as an input. Another useful function class besides 1-#LIN 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 because the other cases are easily shown. For any two 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 each input 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 counting 1TM N that behaves as follows. 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 output tape. Next, N simulates M_f on the input $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$. 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 belongs to 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. □

By 1GAF$_{rat}$, we denote the set of all acceptance functions of rational 1GPFAs. Lemma 7.2 implies that 1-GapLIN is a proper subset of 1GAF$_{rat}$.

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

Proof. The inequality 1-GapLIN $\neq$ 1GAF$_{rat}$ is obvious since certain functions in 1GAF$_{rat}$ can output non-integer values whereas 1-GapLIN contains only integer-valued functions.

For inclusion, we first note that 1-#LIN $\subseteq$ 1GAF$_{rat}$, by re-defining the value $P(u; \sigma; v)$ in the proof of Lemma 6.5 to be the number of accepting computation paths instead of probabilities. By Lemma 7.2, we can write any given function in 1-GapLIN as a difference $f_1 - f_2$ of two functions f_1 and f_2 in 1-#LIN. The desired inclusion now follows from the fact that $1\mathrm{GAF}_{rat}$ is closed under difference, in fact under any linear combinations [29]. $\Box$

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 fall into GSL_{rat} with the cut point 0. Since $\mathrm{GSL}_{rat} = \mathrm{SL}_{rat}$, Lemma 6.4 ensures that A is indeed in 1-PLIN.

Conversely, let A be any language in 1-PLIN. By Theorem 6.3, A is also in SL_{rat}. Following the proof of Lemma 6.4, we can recognize A in linear time by a certain synchronous probabilistic 1TM M which tosses the equal number of fair coins on all computation paths on each input. Let N be the machine obtained from M by exchanging the roles of q_{acc} and q_{rej}. Now, we view M and N as counting 1TMs. Define f and g to be the functions computed by the counting machines M and N, respectively. It follows from the definition that, for every string x, $x \in A$ if and only if $f(x) > g(x)$. Since $f(x) > g(x)$ is equivalent to $(f - g)(x) > 0$, we obtain the characterization $A = \{x \mid (f - g)(x) > 0\}$. Since $f - g$ is in 1-GapLIN by Lemma 7.2, this completes the proof. $\Box$

In comparison, 1-NLIN can be characterized in terms of 1-#LIN as 1-NLIN $= \{A \mid \exists f \in$ 1-#LIN $[A = \{x \mid f(x) > 0\}]\}$.

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 $\subsetneq$ 1-#LIN.

Proof. It is enough 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 $f, g \in$ 1-#LIN for which $A = \{x \mid f(x) > g(x)\}$. By our assumption, these functions fall into 1-FLIN. Using deterministic 1TMs that compute f and g in linear time, we can produce $\left[\begin{smallmatrix} f(x) \\ g(x) \end{smallmatrix}\right]$ in binary in linear time from x. We can further determine whether $f(x) > g(x)$ by comparing $f(x)$ and $g(x)$ bitwise. This gives a deterministic linear-time 1TM for A, and thus A belongs to 1-DLIN. Therefore, we obtain 1-DLIN = 1-PLIN, as requested. $\Box$

7.2 Counting Complexity Classes of Languages

The function classes 1-#LIN and 1-GapLIN naturally induce quite 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 $\{x \in \Sigma^* \mid f(x) \equiv 1 \pmod 2\}$ for certain functions f in 1-#LIN. Obviously, REG $\subseteq$ 1-SPLIN $\subseteq$ 1-$\oplus$LIN. More generally, for each integer $k \geq 2$ and each nonempty proper subset R of $[0, k-1]_{\mathbb{Z}}$, we define the counting class 1-$\mathrm{MOD_{k,R}}$LIN to include all languages of the form $\{x \in \Sigma^* \mid \exists r \in R\ [f(x) \equiv r \pmod k]\}$ for certain functions $f \in$ 1-#LIN. It follows that REG $\subseteq$ 1-$\mathrm{MOD_{k,R}}$LIN and, in particular, 1-$\oplus$LIN = 1-$\mathrm{MOD}_{2,\{1\}}$LIN = co-1-$\mathrm{MOD}_{2,\{0\}}$LIN.

Despite their complex definitions, these counting classes are no more powerful than REG. Hereafter, we wish to 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 = 1-$\mathrm{MOD_{k,R}}$LIN *for every integer* $k \geq 2$ *and every nonempty proper subset* R *of* $[0, k-1]_{\mathbb{Z}}$.

Proof. It suffices to show that 1-$\mathrm{MOD_{k,R}}$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 the sum of probabilities. Recall that $\#M(u)$ denotes the outcome (i.e., the number of accepting paths) of M on input u. For every pair x, z with $|xz| \leq n$, it holds that $\#M(xz) = \sum_{v \in S_n} w_l(x|v) w_r(v|z)$.

Now, let $\mathrm{Supp}_n(x)$ be the set $\{(r, v) \in [0, k-1]_{\mathbb{Z}} \times S_n \mid w_l(x|v) \equiv r \pmod k\}$. We wish to 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. Note that, for each $v \in S_n$, there exists a unique number $r \in [0, k-1]_{\mathbb{Z}}$ satisfying that $(r, v) \in \mathrm{Supp}_n(x)$. This implies that $\#M(xz) - \#M(yz) = \sum_{v \in S_n}(w_l(x|v) - w_l(y|v)) \cdot w_r(v|z) = \sum_{r=0}^{k-1}\sum_{(r,v)\in\mathrm{Supp}_n(x)}(w_l(x|v) - w_l(y|v)) \cdot w_r(v|z)$. For each $(r, v) \in \mathrm{Supp}_n(x)$, we have $w_l(x|v) \equiv r \pmod{k}$ and $w_l(y|v) \equiv r \pmod{k}$ since $\mathrm{Supp}_n(y) = \mathrm{Supp}_n(x)$. It therefore follows that $w_l(x|v) - w_l(y|v) \equiv 0 \pmod{k}$. From this, we conclude that $\#M(xz) - \#M(yz) \equiv 0 \pmod{k}$. Since $\#M(xz) \equiv r_0 \pmod{k}$ for a certain number $r_0 \in R$, we have $\#M(yz) \equiv r_0 \pmod{k}$ for the same r_0. This means that $yz \in L$.

Notice that $N_L(n)$ is bounded above by the number of distinct sets $\mathrm{Supp}_n(x)$ for all strings $x \in \Sigma^{\leq n}$. As a consequence, $N_L(n)$ is upper-bounded by $2^{k|S_n|}$, which is bounded above by a certain constant. Therefore, L belongs to REG. □

We wish to 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 $\text{1-\#LIN}_m^A$ relative to set 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 every input x, M produces an output y_p along each computation path p and $f(x)$ equals the number of all computation 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 complexity 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 $\mathrm{low}_m\mathcal{C}$ to denote the collection of all languages that are many-one low for $\mathcal{C}$. For instance, we obtain $\mathrm{low}_m\text{1-NLIN} = \mathrm{REG}$ since $\text{1-NLIN}_m^{\mathrm{REG}} = \text{1-NLIN}$.

Corollary 7.7 $\mathrm{REG} = \mathrm{low}_m\text{1-\#LIN} = \mathrm{low}_m\text{1-GapLIN}$.

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

We further introduce another counting class $\text{1-C}_=\text{LIN}$ (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 $\text{1-C}_=\text{LIN}$ properly contains REG because the non-regular language $L_{eq} = \{0^n1^n \mid n \in \mathbb{N}\}$ clearly belongs to $\text{1-C}_=\text{LIN}$. Using the closure property of 1-GapLIN, we can easily show that $\text{1-C}_=\text{LIN}$ 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 $f, g \in \text{1-GapLIN}$. 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 $\text{1-C}_=\text{LIN}$.

We wish to show robustness of the complexity class $\text{1-C}_=\text{LIN}$. For comparison, we introduce $\text{1-synC}_=\text{LIN}$ as the subset of $\text{1-C}_=\text{LIN}$ defined by linear-time *synchronous* counting 1TMs. A similar argument to the proof of Lemma 6.5 yields the simple containment $\text{1-C}_=\text{LIN} \subseteq \mathrm{SL}_{rat}^{=}$. Moreover, similar to Lemma 6.4, we can prove that $\mathrm{SL}_{rat}^{=} \subseteq \text{1-synC}_=\text{LIN}$. Therefore, we obtain the following characterization of $\mathrm{SL}_{rat}^{=}$.

Theorem 7.8 $\text{1-C}_=\text{LIN} = \text{1-synC}_=\text{LIN} = \mathrm{SL}_{rat}^{=}$

Earlier, Turakainen [39] proved that SL_{rat} is closed under complementation and that $\mathrm{SL}_{rat}^{=}$ is properly included in SL_{rat}. Symmetrically, $\text{co-SL}_{rat}^{=}$ is also properly included in SL_{rat}. In addition, Dieu [10] showed that $\mathrm{SL}_{rat}^{=}$ is not closed under complementation. Dieu's argument can also work to show that the language $L_{\geq} = \{a^mb^n \mid m \geq n\}$ cannot belong to $\mathrm{SL}_{rat}^{=} \cup \text{co-SL}_{rat}^{=}$. Since $L_{\geq}$ belongs to 1-PLIN, Theorem 7.8 immediately leads to the following separation results.

Corollary 7.9 $\text{1-C}_=\text{LIN} \nsubseteq \text{co-1-C}_=\text{LIN}$, $\text{co-1-C}_=\text{LIN} \nsubseteq \text{1-C}_=\text{LIN}$, *and* $\text{1-C}_=\text{LIN} \cup \text{co-1-C}_=\text{LIN} \subsetneqq \text{1-PLIN}$.

In the next lemma, we briefly summarize basic relationships between $\text{1-C}_=\text{LIN}$ and 1-PLIN.

Lemma 7.10 $\text{1-C}_=\text{LIN} \subseteq \text{1-PLIN} \subseteq \text{1-NLIN}_m^{\text{1-C}_=\text{LIN}} = \text{1-NLIN}_m^{\text{1-PLIN}}$.

Proof. Using Theorems 6.3 and 7.8, the well-known inclusion $\mathrm{SL}_{rat}^{=} \subseteq \mathrm{SL}_{rat}$ yields the first containment $\text{1-C}_=\text{LIN} \subseteq \text{1-PLIN}$. Next, we want to show that $\text{1-PLIN} \subseteq \text{1-NLIN}_m^{\text{1-C}_=\text{LIN}}$. Let A be any set in 1-PLIN. By Proposition 7.4, choose a gap function $f \in \text{1-GapLIN}$ 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 $-2^{k|x|} < f(x) < 2^{k|x|}$.

For brevity, write Δ_k for the set $\{0,1\}^k$ and assume a standard lexicographic order on the set $(\Delta_k)^*$ of strings. Let us define a reduction machine M as follows. On input x, guess a string, say s, over the alphabet Δ_k of length $|x|$ and produce $[\,{}^{x}_{s}\,]$ on the output tape by entering an accepting state. Note that there are exactly $2^{k|x|}$ nondeterministic branches. Note that the machine M is meant to guess the value $f(x)+1$, if $f(x) > 0$, and transfer this information to another machine N'. For each string $s \in (\Delta_k)^{|x|}$, let l_s denote the positive integer l for which s is lexicographically the lth string in $(\Delta_k)^{|x|}$. Obviously, we have $1 \leq l_s \leq 2^{k|x|}$.

Next, we describe the counting 1TM N'. On input $[\,{}^{x}_{s}\,]$, N' guesses a string s' in $(\Delta_k)^{|x|}$ in the third track. In case where $x = \lambda$, N' rejects the input immediately and halt because $\lambda \notin A$. Hereafter, we assume that $x \neq \lambda$. If $s < s'$, then N' produces both an accepting path and a rejecting path, making no contribution to the gap function witnessed by N'. Consider the case where $s' = s$. In this case, N' simulates N on the input x. At length, when $s' < s$, N' rejects the input immediately and halt. Note that $l_s = |\{s' \in (\Delta_k)^{|x|} \mid s' < s\}| + 1$. Let g be the gap function induced by N'. For any nonempty string x and any string $s \in (\Delta_k)^{|x|}$, we obtain $g([\,{}^{x}_{s}\,]) = f(x) - (i_s - 1)$.

With the gap function g, we define a set $B = \{x \mid g(x) = 0\}$, which is in 1-C$_=$LIN. For each string x, if $f(x) > 0$, then $g([\,{}^{x}_{s}\,]) = 0$ for the $f(x)+1$st string s; otherwise, since $f(x) < 0$, $g([\,{}^{x}_{s}\,])$ is always negative for any choice $s \in (\Delta_k)^{|x|}$. This shows that A is many-one 1-NLIN-reducible to B via M; namely, A is in 1-NLIN$_m^B$. Therefore, A belongs to 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$.

Finally, we want to show that 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$ = 1-NLIN$_m^{\text{1-PLIN}}$. 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$_m^{\text{1-C}_=\text{LIN}}$, 1-NLIN$_m^{\text{1-PLIN}}$ is contained in the complexity class 1-NLIN$_m^{\text{1-NLIN}_m^{\text{1-C}_=\text{LIN}}}$, which coincides with 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$ by Proposition 4.5. □

The next proposition demonstrates two separation results concerning two complexity classes 1-C$_=$LIN and 1-PLIN. Notationally, low$_m$1-PLIN denotes the complexity class that is many-one low for 1-PLIN (i.e., low$_m$1-PLIN = $\{A \mid$ 1-PLIN$_m^A \subseteq$ 1-PLIN$\}$).

Proposition 7.11 *1.* 1-PLIN $\subsetneqq$ 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$ $\cap$ co-1-NLIN$_m^{\text{1-C}_=\text{LIN}}$.

2. low$_m$1-PLIN $\subsetneqq$ 1-PLIN $\subsetneqq$ 1-PLIN$_m^{\text{1-PLIN}}$.

Proposition 7.11 is a consequence of the following key lemma regarding the complexity of the language *Center* $= \{x1y \mid x,y \in \{0,1\}^*, |x| = |y|\}$.

Lemma 7.12 *The language Center belongs to* 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$ $\cap$ co-1-NLIN$_m^{\text{1-C}_=\text{LIN}}$.

With help of this lemma, we can prove Proposition 7.11 easily.

Proof of Proposition 7.11. We have seen in Section 6 that *Center* $\notin \mathrm{SL}_{rat}$. Assertion 1 follows directly from Lemma 7.12. The second part of Assertion 2 follows from Assertion 1 because 1-NLIN$_m^A \subseteq$ 1-PLIN$_m^A$ for any set A. Next, we want to show the first part of Assertion 2. Let A be any set in low$_m$1-PLIN, that is, 1-PLIN$_m^A \subseteq$ 1-PLIN. Obviously, $A \leq_m^{\text{1-PLIN}} A$, and thus $A \in$ 1-PLIN$_m^A$. This implies that $A \in$ 1-PLIN$_m^A \subseteq$ 1-PLIN. Therefore, we obtain low$_m$1-PLIN $\subseteq$ 1-PLIN. The separation low$_m$1-PLIN $\neq$ 1-PLIN follows from the second part of Assertion 2 because low$_m$1-PLIN = 1-PLIN implies 1-PLIN = 1-PLIN$_m^{\text{1-PLIN}}$ by the definition of lowness. □

To complete the proof of Proposition 7.11, we still need to prove Lemma 7.12. The lemma can be proven by constructing many-one 1-NLIN-reductions from *Center* to sets in 1-C$_=$LIN.

Proof of Lemma 7.12. As a target set, we use the set $A = \{0^n\#1^n \mid n \in \mathbb{N}\}$, where # is a special symbol not in $\{0,1\}$, which belongs to 1-C$_=$LIN. First, we want to show that *Center* $\leq_m^{\text{1-NLIN}} A$, since this implies that *Center* $\in$ 1-NLIN$_m^{\text{1-C}_=\text{LIN}}$. Let us consider the following nondeterministic 1TM N running in linear time.

> Let x be any input. In Phase 1, determine whether $|x|$ is odd and then return the head back to the start cell. If $|x|$ is even, output x and halt immediately. Now, assume that $|x|$ is odd. In Phase 2, choose nondeterministically either 0 or 1. If 1 is chosen, go to Phase 3; otherwise, overwrite 0 in the scanning cell, move the head to the right, and then repeat Phase 2. Whenever the head reaches

> the first blank symbol, return it to the start cell and halt. In Phase 3, check whether the head is currently scanning 1. If not, return the head back to the start cell and halt. Otherwise, change 1 to # and then, by moving the head rightward, convert all the symbols on the right of # to 1s. Finally, return the head to the start cell and halt.

We now prove that $Center \leq_m^{\text{1-NLIN}} A$. Assuming that $x \in Center$, let $x = u1v$ for two strings u and v of the same length. Along a certain computation path, N successfully converts x to $0^{|u|}\#1^{|v|}$, which belongs to A. On the contrary, assume that $x \notin Center$. When $|x|$ is even, N outputs x, which is obviously not in A. In case where x is of the form $u0v$ with strings u and v of the same length, N never outputs $0^{|u|}\#1^{|v|}$. Hence, N many-one 1-NLIN-reduces $Center$ to A.

To show that $\overline{Center} \leq_m^{\text{1-NLIN}} A$, let us consider another nondeterministic 1TM N' that behaves as follows.

> On input x, check if $|x|$ is odd. If not, output 1#1 and halt. Otherwise, simulate Phase 2 of N's algorithm. In Phase 3, check if the currently scanning cell has 1. If so, return the head to the start cell and halt. Otherwise, overwrite # and convert the whole input x into a string of the form $0^n\#1^m$ as an output.

A similar argument for N can demonstrate that N' many-one 1-NLIN-reduces $\overline{Center}$ to A. Hence, we conclude that $Center \in \text{co-1-NLIN}_m^{\text{1-C}_=\text{LIN}}$. This completes the proof. □

The complexity class that is many-one low for 1-C$_=$LIN, denoted low$_m$1-C$_=$LIN, satisfies the inclusion relations low$_m$1-C$_=$LIN $\subseteq$ 1-C$_=$LIN $\cap$ co-1-C$_=$LIN $\subsetneqq$ 1-C$_=$LIN. The first inclusion can be proven in a similar fashion to the proof of Corollary 7.7 and the second proper inclusion follows from Corollary 7.9. Unlike Corollary 7.7, it is open whether REG = low$_m$1-C$_=$LIN.

Lemma 7.13 low$_m$1-C$_=$LIN $\subseteq$ 1-C$_=$LIN $\cap$ co-1-C$_=$LIN $\subsetneqq$ 1-C$_=$LIN.

Since 1-C$_=$LIN $\subseteq$ 1-PLIN, Proposition 6.7 immediately yields the following separations: CFL $\cap$ REG/$n \nsubseteq$ 1-C$_=$LIN, CFL $\nsubseteq$ 1-C$_=$LIN$\cap$REG/n, and REG/$n \nsubseteq$ CFL$\cup$1-C$_=$LIN. Other separations among three complexity classes CFL, 1-C$_=$LIN, and REG/n are presented in the following proposition.

Proposition 7.14 *1-C$_=$LIN $\cap$ CFL $\nsubseteq$ REG/n, 1-C$_=$LIN $\cap$ REG/n $\nsubseteq$ CFL, and 1-C$_=$LIN $\nsubseteq$ REG/$n \cup$ CFL.*

Proof. Let us consider the language $Equal = \{w \in \{0,1\}^* \mid \#_0(w) = \#_1(w)\}$, which stays outside of REG/n. The first separation follows immediately since $Equal$ belongs to CFL and 1-C$_=$LIN. To prove the second claim, recall the non-context-free language $L_{3eq} = \{a^nb^nc^n \mid n \in \mathbb{N}\}$. We want to show that L_{3eq} belongs to 1-C$_=$LIN. To see this, note that $L_1 = \{a^nb^nc^m \mid m,n \in \mathbb{N}\}$ and $L_2 = \{a^mb^nc^n \mid m,n \in \mathbb{N}\}$ are in 1-C$_=$LIN. It is rather easy to show that the intersection $L_1 \cap L_2$ is also in 1-C$_=$LIN. Since $L_{3eq} = L_1 \cap L_2$, L_{3eq} belongs to 1-C$_=$LIN.

The second separation follows from the fact that $L_{3eq} \in$ REG/n. For the third separation, consider the non-context-free language *3Equal*, given in Section 6. It is straightforward to show that *3Equal* belongs to 1-C$_=$LIN. With a similar argument for the first separation, we can argue that *3Equal* cannot be in REG/n. □

8 Quantum Computation

The notion of a quantum Turing machine was introduced by Deutsch [9] in the mid 1980s and later reformulated by Bernstein and Vazirani [5] to model a quantum computation. Within our framework of 1TMs, we use a general model of one-tape quantum Turing machines, which allow their tape heads to stay still [45, 46].

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. Any element of this configuration space is called a *superposition of configurations*, which is a linear combination of configurations with complex coefficients (called *amplitudes*). A 1QTM M is said to be *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 another superposition of the configurations resulting 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 viewed 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 1QTM. In accordance with 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 quantum computation of M on the input x, all configurations in T become halting configurations at time t *for the first time*. If such a t exists, we say that M *halts*[‖] *at time* t. This halting condition makes us view time-bounded 1QTMs as classical "synchronous" machines. A time-bounded 1QTM M is said to be *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 quantum 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., configurations with the internal state q_{acc}) in any superposition generated at the time when M halts on the 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 [26] mainly because of the synchronous condition of our 1QTMs. On the contrary, the synchronous condition enables us to prove in Lemma 8.1 a strong connection between 1QTMs and 1-GapLIN.

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 [46] (see also [15]). In the following proof, we use the folding machine obtained from a given 1QTM.

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, Since the construction of a folding machine, given in Section 4.2, is applicable to any 1QTM, we can work on M's 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 violate unitarity and no longer be 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 (classical) computation path of N on input x. When N halts, an accepting configuration of N depends only on the tape content because N's internal 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 only the contents of cells in the input area. We denote by $amp_N(x, y, z)$ the amplitude associated with accepting computation path y of N on input x leading to the final tape content z. Since N is synchronous, the value $amp_N(x, y, z) \cdot c^{\mathrm{Time}_M(x)}$ is always an integer.

Now, we define the function f_+ as $f_+([\,{x \atop z}\,]) = c^{\mathrm{Time}_M(x)} \cdot \sum_y amp_N(x, y, z)$, 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 want to show that $f \in$ 1-FLIN. For our purpose, we first translate the 1QTM N into a 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 exactly $|m|$ 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 leading to certain accepting configurations. To determine the sign of the amplitude associated with each computation path of N, we further prepare two sets of internal states for $\hat{N}$ and move one set to another whenever the amplitude sign changes by an application of δ. The resulting machine witnesses that f_+ is in 1-#LIN. Similarly, we define $f_-([\,{x \atop z}\,]) = c^{\mathrm{Time}_M(x)} \cdot \sum'_y amp_N(x, y, z)$, 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 conclude that $f_- \in$ 1-#LIN.

Recall that the acceptance probability $p_M(x)$ is 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, $c^{2\mathrm{Time}_M(x)} \cdot p_M(x) = \sum_{z \in \Gamma^{|x|}} (f_+([\,{x \atop z}\,]) - f_-([\,{x \atop z}\,]))^2$.

[‖] This definition comes from Bernstein and Vazirani [5], who defined a quantum Turing machine to "halt" at time t if the superposition of configurations at time t consists only of halting configurations and, at time less than t, the superposition contains no halting configuration. See also [45, 46] for more discussions.

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, we set $d = c^2$. This completes the proof. □

A simple application of Lemma 8.1 shows the following proposition.

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 1QTM with $\mathbb{Q}$-amplitudes, which produces no computational error. From this, it follows that 1-revDLIN $\subseteq$ 1-BQLIN$_{\mathbb{Q}}$. Proposition 3.2 therefore implies that REG $\subseteq$ 1-BQLIN$_{\mathbb{Q}}$.

We want to show the second inclusion. Let L be any language in 1-BQLIN$_{\mathbb{Q}}$ and choose 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 (by, e.g., a majority vote technique), 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 string x. It thus follows that $x \in L$ implies $3f(x) > 2g(x)$ and that $x \notin L$ 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. This result can be compared with a polynomial-time case of NQP$_{\mathbb{C}}$ = co-C$_=$P [47].

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 U and V 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 $V|s\rangle = \frac{(1-s)4}{5}|s\rangle + \frac{3}{5}|(s+2) \bmod 4\rangle$ for $s \in \{0,2\}$ and $V|s\rangle = \frac{3}{5}|s\rangle + \frac{(s-2)4}{5}|(s+2) \bmod 4\rangle$ 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 certain 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) one 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.

Hereafter, we wish to construct a linear-time well-formed stationary 1QTM M with $\{0, \pm 3/5, \pm 4/5, \pm 1\}$-amplitudes and show that, for any nonempty string x, $p_M(x) > 0$ if and only if $p_N(x) \neq 1/2$. From this, we can conclude that L belongs to 1-NQLIN$_{\mathbb{Q}}$. Let $x = \sigma_1 \ldots \sigma_n$ be any string, where each symbol σ_j is in Σ and $n \geq 0$. Let $\Delta = \{0,1\}^m$ be our new alphabet. Assuming a linear order on Δ, for each symbol $k \in \Delta$, we define l_k to be the number satisfying that k is the $l_k + 1$st symbol in Δ. Note that $0 \leq l_k < 2^m$ for any $k \in \Delta$.

1) Initially, M is in the initial state q_0, scanning the start cell (indexed 0). If $x = \lambda$, then M immediately accepts or rejects the input if $\lambda \in L$ or $\lambda \notin L$, respectively. In the rest of the description of M, we assume that $|x| \geq 1$. In this preprocessing phase, M replaces each input symbol σ by its corresponding new symbol $\left[\begin{smallmatrix} \sigma \\ 0^m \end{smallmatrix} \right]$

by moving its tape head rightward. In the end, M returns the tape head to the start cell. In the subsequent description of M, we pay our attention to the content of the cells indexed between 0 and n.

2) The machine M simulates a series of "coin flips" of N by generating a certain superposition of configurations. By moving the tape head rightward again, M applies the transformation $U^{\otimes m}$ to the symbol 0^m given in $\left[\begin{smallmatrix} \sigma \\ 0^m \end{smallmatrix}\right]$: $U^{\otimes m}|0^m\rangle = \sum_{k\in\Delta} \left(\frac{3}{5}\right)^{\#_0(k)} \left(\frac{4}{5}\right)^{\#_1(k)} |k\rangle$, where $\#_i(k)$ denotes the number of i's in k when k is viewed as an m-bit string. When M reaches the first blank symbol (in the nth cell), it returns the tape head to the start cell. On each (classical) computation path, the tape content must be in the form $\left[\begin{smallmatrix} x \\ \vec{k} \end{smallmatrix}\right] = \left[\begin{smallmatrix} \sigma_1 \\ k_1 \end{smallmatrix}\right]\left[\begin{smallmatrix} \sigma_2 \\ k_2 \end{smallmatrix}\right]\cdots\left[\begin{smallmatrix} \sigma_n \\ k_n \end{smallmatrix}\right]$, where $\vec{k} = k_1k_2\cdots k_n \in \Delta^n$.

3) Assume that N's initial state is 0. Let p_0 be a new internal state of M associated with N's. Now, we make M simulate each step of N in such a way that, when N makes a transition from an internal state a to another state b with transition probability $r_{a,b}$ for an input symbol σ, M generates exactly $r_{a,b}$ (classical) computation paths. This can be done with new internal states p_a and p_b and by applying the following transition rule: for every symbol $k \in \Delta$, $\delta(p_a, \left[\begin{smallmatrix} \sigma \\ k \end{smallmatrix}\right]) = |p_1\rangle|\left[\begin{smallmatrix} \sigma \\ k \end{smallmatrix}\right]\rangle|R\rangle$ if $0 \leq l_k < r_{a,1}(\sigma)$ and $\delta(p_a, \left[\begin{smallmatrix} \sigma \\ k \end{smallmatrix}\right]) = |p_b\rangle|\left[\begin{smallmatrix} \sigma \\ k \end{smallmatrix}\right]\rangle|R\rangle$ if $b > 1$ and $\sum_{i=1}^{b-1} r_{a,i}(\sigma) \leq l_k < \sum_{i=1}^{b} r_{a,i}(\sigma)$.

4) After reaching the nth cell in internal state p_a, M writes down the outcome 0 or 1 of the N depending on $a \in F$ or $a \notin F$, respectively, and then enters a new internal state s_a. When $\left[\begin{smallmatrix} x \\ \vec{k} \end{smallmatrix}\right]$ is produced in phase 2, let $r_{\vec{k}}$ denote this outcome written in the nth cell. Note that $p_N(x) = |\{\vec{k} \in \Delta^n \mid r_{\vec{k}} = 0\}| \cdot 2^{-mn}$.

5) In this phase, M first reverses phase 3. This brings the tape head back to the start cell and the internal state to p_0. By moving the head rightward again, M also applies $U^{\otimes m}$ to each symbol k in $\left[\begin{smallmatrix} \sigma \\ k \end{smallmatrix}\right]$, just as in phase 2, collapsing at most 2^m branches to each configuration containing tape content $\left[\begin{smallmatrix} x \\ \vec{k} \end{smallmatrix}\right] r$, where $\vec{k} \in \Delta^n$ and $r \in \{0, 1\}$. In particular, we obtain the configuration with $\left[\begin{smallmatrix} x \\ 1^{mn} \end{smallmatrix}\right] r = \left[\begin{smallmatrix} \sigma_1 \\ 1^m \end{smallmatrix}\right]\left[\begin{smallmatrix} \sigma_2 \\ 1^m \end{smallmatrix}\right]\cdots\left[\begin{smallmatrix} \sigma_n \\ 1^m \end{smallmatrix}\right] r$ with amplitude $amp(r) = \left(\frac{12}{25}\right)^{mn} |\{\vec{k} \in \Delta^n \mid r_{\vec{k}} = r\}|$.

6) 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.

7) Finally, N checks if the cells indexed between 0 and n consists of $\left[\begin{smallmatrix} x \\ 1^{mn} \end{smallmatrix}\right] 0$. If so, M enters q_{acc}; otherwise, M enters q_{rej}. This phase can be done in a reversible fashion. This completes the description of M.

For any nonempty string x, a simple calculation shows that the acceptance probability $p_M(x)$ of M is $p_M(x) = \left(\frac{12}{25}\right)^2 (amp(0) - amp(1))^2$, which equals $\left(\frac{24}{25}\right)^{2mn+2} (p_N(x) - 1/2)^2$. It therefore follows that $p_M(x) > 0$ if and only if $p_N(x) \neq 1/2$, as requested. □

9 Epilogue

By exploring the close relationships to 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 in power, 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 further research on the theory of one-tape linear-time Turing machines, we suggest five important future directions of the research.

i) The model of Turing machines has significantly evolved over the past four decades. We have shown in this paper that different machine types can alter the power of computation. There are many more machine types that we have not yet discussed in this paper. Other types of Turing machines include metric Turing machines, bottleneck Turing machines, and interactive Turing machines (see, e.g., [11, 17, 27]). 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 associated finite state automata. To study the power of Turing machines, we need to explore their special ability to compute "functions" instead. In the past, such functions have been studied extensively in terms of *search problems*, *optimization problems*, and *approximation problems*. The study of these 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 linear-time complexity classes that we have studied in this paper are unlikely to possess "complete" problems via many-one 1-DLIN-reductions. Is there any "weak" reducibility that highlights the relative complexity of languages?

iv) We have considered advised computations; however, the role of advice has not been 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 types of meaningful relativization is also necessary for one-tape linear-time complexity classes.

We hope that the further study of the above structural complexity issues on resource-bounded computations will lead to the better understandings of the effect of bounded resources of Turing machines

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. The authors gracefully appreciate critical comments of anonymous referees.

References

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

[2] T. P. Baker, J. Gill, and R. Solovay. Relativizations of the P =? NP question. *SIAM Journal on Computing*, **4** (1975), 431–442.

[3] J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity II*, Springer-Verlag, 1990.

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

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

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

[7] 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.

[8] 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.

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

[10] P. D. Dieu. On a class of stochastic languages. *Zietschr. f. math. Logik und Grundlagen d. Math. Bd.*, **17** (1971), 421–425.

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

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

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

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

[15] L. Fortnow and J. Rogers. Complexity limitations on quantum computation. *Journal of Computer and System Sciences* **59** (1997), 202–209.

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

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

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

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

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

[21] J. Kaņeps. Stochasticity of the languages recognizable by 2-way finite probabilistic automata. *Diskret. Mat.* **1** (1989), 63–77 (in Russian).

[22] J. Kaņeps 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.

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

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

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

[26] 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.

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

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

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

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

[31] M. Nasu and N. Honda. A context-free language which is not acceptable by a probabilistic automaton. *Inform. Control*, **18** (1971), 233–236.

[32] 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.

[33] W. J. Paul, E. J. Prauß, R. Reischuk. On Alternation, *Acta Informatica*, **14** (1980), 243-255.

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

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

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

[37] A. L. Selman. A taxonomy of complexity classes of functions. *Journal of Computer and System Sciences*, **48** (1992), 357–381.

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

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

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

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

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

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

[44] 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.

[45] 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.

[46] 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.

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

[48] 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 internal states of M. By its definition, M has at least three states (that is, $q \geq 3$). Choose a number $n_0 \in \mathbb{N}$ such that $T(n) > 0$ for all numbers $n \geq n_0$. We define the function f from $\{n \in \mathbb{N} \mid n \geq n_0\}$ to $\mathbb{R}^{\geq 0}$ by $f(n) = \sqrt{n \log n / T(n)}$. Since $T(n) = o(n \log n)$, it follows that $\lim_{n\to\infty} f(n) = \infty$. Choose the smallest number $c \in \mathbb{N}$ such that, for every $n \geq 2$,

$$\frac{3\left(q^{\frac{\log n}{f(n)}+1} - 1\right)}{q-1} \leq n\left(1 - \frac{1}{f(n)}\right) + \frac{c \cdot f(n)}{\log n} + 1.$$

Such c exists because $q^{\frac{\log n}{f(n)}+1} = o(n)$.

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

Let x_0 denote lexicographically the first input string that has a long crossing sequence. Let $n_0 = |x_0|$. Let s_0 be the shortest computation path of M on the 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, along the computation path s_0.

Let us consider all critical boundaries of x_0 along the path s_0 whose crossing sequences are of lengths at most $\log n_0 / f(n_0)$. Let h be the number of all such critical boundaries. Since the total computation steps along the path s_0 is equal to the sum of the lengths of any crossing sequences at intercell boundaries, we have $T(n_0) > c + (n_0 + 1 - h)\frac{\log n_0}{f(n_0)}$. The inequality comes from the assumption that the length of γ_0 is longer than c. Thus, we have

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

which is at least the number of all crossing sequences of lengths at most $\log n_0 / f(n_0)$. Hence, there exist at least four distinct critical boundaries b_1, b_2, b_3, b_4 that have an identical crossing sequence in the path s_0. Clearly, 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 a valid computation path of M on the input x_0' and still has a crossing sequence whose length is greater than c. This contradicts the minimality of x_0. Therefore, the lemma holds. □

Proof of Lemma 4.3. Let n be any number in $\mathbb{N}$. For each string $x \in \Sigma^{\leq n}$ and each crossing sequence $v \in S_n$, 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 the intercell boundary between x and z along a certain *accepting* computation path of M on the input xz. Now, let $\mathrm{Supp}_n(x) = \{v \in S_n \mid\ x\ n\text{-supports } v\ \}$.

We want to show that, for any three strings $x, y, z \in \Sigma^*$, 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 any crossing sequence between x and z along a certain accepting computation path of M on the input xz. Clearly, we have $v \in \mathrm{Supp}_n(x)$. 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 the input yz. By the nature of crossing sequences, yz has an accepting computation. Thus, we conclude that $yz \in L$. Similarly, we obtain the same conclusion in the case where the tape head halts in the right region of v.

Note that $N_L(n)$ should be at most the number of distinct sets $\mathrm{Supp}_n(x)$ over all strings $x \in \Sigma^{\leq n}$. Therefore, $N_L(n)$ is upper-bounded by $2^{|S_n|}$. □