INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Structural Properties of One Tape Turing Machines

by

Jack Cheng-Hung Lin

Thesis
Submitted to the School of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Masters in Computer Science

Under the supervision of Prof. Tomoyuki Yamakami

School of Information Technology and Engineering
University of Ottawa
Ontario. Canada

October 23. 2002

Abstract

The model of Turing machines has been studied since its birth in 1936. Researchers have continuously proposed variants of such a model. Upon imposing different constraints, the power of each model varies or even remains the same, accordingly. Some well-known result (for example, the equivalence of finite state automata and one-tape linear-time deterministic Turing machines) has proven that the abilities of overwriting the tape content and scanning the tape content more than once cannot gain any advantage under certain restrictions.

In this thesis, we study the behaviors and the fundamental properties of variants of one-tape Turing machines, such as deterministic, reversible, nondeterministic, probabilistic, and quantum Turing machines. This gives us a better understanding about the strength and the weakness of each machine type. For example, under the one-tape linear-time restriction, reversible, nondeterministic, co-nondeterministic, and bounded-error probabilistic computations recognize exactly regular languages whereas probabilistic and quantum Turing machines can recognize even non-regular languages.

ACKNOWLEDGMENT

I would like to first thank Dr.Tomoyuki Yamakami (University of Ottawa) for his supervision, especially useful comments and advice during the last two years. Secondly, I would like to thank Dr.Toshio Suzuki (Osaka Prefecture University) for his lectures in logics and Dr.Kohtaro Tadaki (Japan Science and Technology Corporation) for all the helps and discussions in complexity issues during their visits in Ottawa. I am also grateful to Dr.Amy Felty (University of Ottawa) and Dr.Dong Howe (Carleton University) for being my committee members as well as for all their precious time spent on reading my thesis and for their constructive comments and suggestions. Furthermore, I would like to thank Franck Binard (University of Ottawa), Robert S. Krukowski (Western University), Dr.Harumichi Nishimura (University of Ottawa), Marina Sokolova (University of Ottawa), and David Steeves (Carleton University) for proof-reading my thesis. Finally, I thank my father, Ching-Kuei Lin; my mother, Yen-Tsai Lin Yen; my sister, Betty Yu-Wen Lin; and my brother, Joe Chih-Yuan Lin for all their supports.

Contents

List of Figures

Introduction

0.1 History

To describe the history of theoretical computer science, one cannot avoid mentioning mathematics. Theoretical computer science have adapted insights, definitions, problems, and techniques from several fields of mathematics such as set theory, graph theory, recursion theory, number theory, and so forth.

Theory of Computation

The history began when logicians first started to think about the computation for mathematical problems. A well-known question is Hilbert's (1862-1943) tenth problem [34] proposed in 1901. The question asks if there exists a process of finite operations to determine whether a polynomial has an integral root. To show that such a process does not exist[1], we must first have a clear definition of a process of operations (called an algorithm). The meaning of "algorithm" was not defined until 1936.

In 1928 Hilbert addressed another question: is mathematics *complete, consistent,* and *decidable?* Completeness means that every mathematical statement can be proved or disproved, consistency means that no valid statement can prove an invalid statement, and decidability means that, upon given a mathematical problem, there exists a mathematical method to apply in order to have an yes/no answer at the end. In 1931 Gödel [32] (1906-1978) gave solutions to the first two sub-questions. In 1936 Turing [70] (1912-1954) solved the third one by defining a machine, known as Turing machine, and showed that some question (for example, the Halting Problem) is not decidable. His machine gave a precise definition to "algorithm." Meanwhile, Church [13] (1903-1995) solely defined algorithm using λ-calculus. Later, both definitions were proven to be equivalent [13, 41]. Independently, a similar mathematical model, Post machine, was proposed by Emil Post [64] (1897-1954). Thereafter, researchers have expanded these ideas and introduced variants of Turing machines.

Furthermore, Kleene's (1909-1994) research on the development of *recursion theory* [43] with Church, Gödel, Turing, and Post have played an important role in the foundations of theoretical computer science. His work provides methods of identifying solvable and unsolvable problems. This lead to the study of computational complexity.

[1] In 1970 Matijasevič [54] showed that such an algorithm does not exist.

Computational Complexity

Several polynomial-time solvable problems were brought to attention by Cobham [14] and Edmonds [24] in the mid 1960's. In addition, Edmonds [25] argued that polynomial-time computations were efficient and gave an informal description of nondeterministic polynomial-time. The class of sets recognized nondeterministically in polynomial time is called **NP**.

In the 1970's Cook [17] and Levin [49] introduced the notion of **NP**-*complete* problems. This has given researchers advantages of targeting only on these problems while trying to solve all **NP** problems. Shortly, Karp [39] showed that many natural combinatorial problems are indeed **NP**-*complete*. Henceforth, many questions have arisen concerning **NP**. Among all, the most well-known headache to researchers is the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem. People have tried to find a cure for such a headache; however, no one has yet succeeded. Until now, many interesting ideas have been shown, many approaches have been taken, many techniques have been created, and theorems have been proven. In 1975 Ladner [48] showed that if **P** is different from **NP**, then there are incomplete sets in **NP**; and in 1982 Mahaney [53] showed that if there exist[2] sparse complete sets for **NP**, then **P** equals **NP**. Moreover, in 1972 Meyer and Stockmeyer [56] defined the polynomial-time hierarchy. Thereafter, separating this hierarchy has become a new approach to solve the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem. An obvious implication is that if $\mathbf{P} = \mathbf{NP}$, then the polynomial-time hierarchy collapses.

Quantum Computational Complexity

The history of quantum computers began with the creation of quantum mechanics. Since the world is quantum mechanical, we might be able to build a machine that follows the rules of quantum physics and benefits from the quantum effect. In the 1980's scientists started to transfer such an idea to computer science. Astonishing results have been shown ever since.

Subsequent to Feynman's [28] idea of "quantum computer" in 1982, its massive parallelism has been attracting much attention. He indicated that the natural simulation of a quantum physical system on a probabilistic Turing machine requires an exponential slowdown. In other words, computing devices using the quantum mechanism might be more powerful than any classical device since a quantum mechanical process may not be simulated efficiently by classical computers.

Quantum information theory requires any quantum computation to be reversible. In classical theory, such a constraint is not necessary. In 1973 Bennett [6] showed the possibility of reversing a deterministic computation. The trick is to record all the information during the computation to a "history" tape and retrieve them later by back-tracking. In 1980 Benioff [5] described a computing device based on quantum mechanism. He extended the earlier work of Bennett and showed that a reversible Turing machine is theoretically possible. If we consider a reversible Turing machine as a dynamic system, then it can be run forward and backward during the entire computation, and there is no loss of information.

In 1985 Deutsch [19] described the first quantum Turing machine and then introduced the

[2] A sparse set is a set that is polynomial in size.

notion of quantum networks [20] in 1989. How fast can a quantum Turing machine simulate another quantum Turing machine? Deutsch showed the existence of a universal quantum Turing machine and questioned whether such a machine needs an exponential slowdown to simulate another quantum Turing machine. This problem had remained open until 1993, when Bernstein and Vazirani [8], and Yao [77] showed that such a simulation required polynomial-time slowdown. Another open question that concerns researchers is the computability of a quantum computer in comparison with classical computers. However, some solid evidence has proven that, in an oracle setting, quantum Turing machines are more powerful than probabilistic Turing machines. This was first proven by Berthiaume and Brassard [9] and improved by Bernstein and Vazirani [8], and Simon [67].

0.2 Goals and Purposes of the Thesis

Although, polynomial-time algorithms are considered to be efficient in theory, it is more practical in reality to restrict resources such as the number of tapes, the number of heads, or even smaller time/space bounds. One practical reason is that even a cubic-time computation would take long in real-life to execute.

Since finite state automata are the simplest mathematical model in the theory of computation, it is reasonable to take this model and improve its computability by allowing more resources and equipping it with more computation powers. The most direct way might be to add to the automata the abilities of writing and to scanning the tape content more than once. To go one step further, we allow such a model with the power of nondeterminism, reversibility, probabilistic coin-flip, and quantum interference and quantum superposition. This has helped us indicate the strength and the weakness of each variant of Turing machines. In general, under the one-tape linear-time restriction, the models with unbiased probabilistic coin-flip and quantum effect have more computational power over the others. Moreover, since, in polynomial time, many problems have remained unsolved, studying more restricted and simpler models would give a better understanding of the fundamental structure and the behavior of each computing device. In addition, studying variants of Turing machines also gives a different perspective on solving problems. A well-known example is the model of alternating Turing machines[3] introduced by Chandra, Kozen, and Stockmeyer [11] in 1981 to characterize the polynomial-time hierarchy.

In this thesis, we study the behaviors of variants of Turing machines: deterministic, reversible, nondeterministic, probabilistic, and quantum Turing machines. We impose the one-tape constraint as well as the time constraints, accordingly. For the time constraints, we mainly focus on the $O(n)$ time, $o(n \log n)$ time, and $O(n \log n)$ time bounds. In general, this thesis is mainly sectioned into three chapters: deterministic and nondeterministic classes, probabilistic and counting classes, and quantum computation.

In Chapter 1, we briefly introduce the concept of formal languages and the theory of

[3] An alternating Turing machine performs in turn nondeterministic computations and co-nondeterministic computations.

computation. This thesis includes most results that appeared in [51] along with some additional concepts, techniques, and results. Chapter 2 consists of three parts: deterministic computation, nondeterministic computation, and set-theoretical difference of two languages. In Section 2.1, we sketch some results, techniques, and ideas.

It is often hard to find a desired method to separate complexity classes. Computer scientists have developed tools such as the crossing sequence (see Section 2.1.3), Kolmogorov complexity (see Section 2.1.4), and communication complexity (see Section 2.2.4). In general, the crossing sequence technique views a computation by observing the ordered sequence of states on a particular inter-cell boundary (a boundary between two adjacent cells in a tape of a Turing machine). Kolmogorov complexity measures the amount of information encapsulated within a given string, and communication complexity measures the number of bits interchanged between two parties. In Section 2.2, we expand several results on deterministic computation to nondeterministic and co-nondeterministic computations. We show the equivalence of the class of regular languages, one-tape $o(n \log n)$-time nondeterministic class, and one-tape $o(n \log n)$-time co-nondeterministic class. This further implies that the one-tape $o(\log n)$-time nondeterministic class is closed under complementation. To show the separation among deterministic, nondeterministic, and co-nondeterministic computations, we relax the time bound from $o(n \log n)$ to $O(n \log n)$. Rather interestingly, we show that the complementation property of one-tape $O(n \log n)$-time nondeterministic class fails. Furthermore, we show that the one-tape $O(n \log n)$-time deterministic class is strictly included in the one-tape $O(n \log n)$-time nondeterministic class and the one-tape $O(n \log n)$-time co-nondeterministic class.

The first half of Chapter 3 deals with the bounded-error probabilistic class, and in the second half of this chapter, a counting class is studied. In Section 3.1, we extend the crossing sequence technique and prove the equivalence of the class of regular languages and the one-tape $o(n \log n)$-time bounded-error probabilistic class. This implies that, under the one-tape $o(n \log n)$-time restriction, any bounded-error probabilistic computation is equivalent to some nondeterministic computation as well as co-nondeterministic computation. In Section 3.2, we first show the existence of a non-regular language and a non-context-free language in a counting class that returns the difference between the number of accepting paths and the number of rejecting paths in a nondeterministic computation. Later in the section, we show its non-closure property on complementation using the proof technique in [22, 59].

Would quantum effect benefit our computation with such a restricted condition? In Chapter 4, we show that the class of regular languages is strictly included in (one-tape linear-time) co-nondeterministic quantum class by defining a quantum algorithm that recognizes a non-regular language under such a restriction. This further implies that, by substituting a quantum Turing machine for a nondeterministic Turing machine, the computability increases, which gives an affirmative answer to our question earlier. In what follows, we show that, under the one-tape linear-time restriction, nondeterministic quantum class is not closed under complementation using the same technique in the previous chapter along with the properties of unitary transformation. In addition, we show that the counting class studied in Chapter 3

is a subset of one-tape linear-time co-nondeterministic quantum class.

0.3 Author's Contribution to This Thesis

Here is the list of the author's contribution to this thesis.

Section 2.2.5: Lemma 2.2.16. Proposition 2.2.17. Theorem 2.2.18. and Theorem 2.2.19.

Section 2.3: Lemma 2.3.2 and Lemma 2.3.3.

Section 3.1.3: the author revises the proof and gives a different method to bound the probability.

Section 3.2.3: Lemma 3.2.5. Proposition 3.2.6. Proposition 3.2.7. and Proposition 3.2.8.

Section 3.2.4: Theorem 3.2.10. Lemma 3.2.11. Theorem 3.2.12. and Theorem 3.2.13.

Section 4.4: Lemmas 4.4.1. Lemma 4.4.4. Corollary 4.4.5. and Lemma 4.4.6.

Section 4.5: Theorem 4.5.1. Lemma 4.5.2. Theorem 4.5.3. and Theorem 4.5.4.

Section 4.6: Theorem 4.6.1.

Chapter 1

Preliminary

In this chapter, we give an introduction to linear algebra, formal language, and the theory of computation. In Section 1.1, we define, in general, some notions and notations in linear algebra. In Section 1.2, we give an introduce to formal language and the theory of computation. More precisely, we give formal definitions of regular languages and context-free languages along with some examples in the first half of this section and some insight about the computations of Turing machines and recursion theory in the remaining section.

1.1 Linear Algebra

1.1.1 Notations

Let $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ be the sets of all integers, nonnegative integers, rational numbers, real numbers, and complex numbers, respectively. In particular, $\mathbb{N}^+$ denotes the set of all positive integers, $\mathbb{R}^{\geq 0}$ denotes the set of all nonnegative real numbers and $\tilde{\mathbb{C}}$ denotes the set of computable complex numbers.

For convention, we denote w, x, y, z to be strings over alphabet Σ. The *lexicographical ordering* is the standard order of basis that can be seen as increments of lengths and integers: $\epsilon < 0 < 1 < 00 < 01 < 10 < 11 < 000 < \ldots$. A set of strings is called a *language* or a *problem*. For simplicity, we use a binary representation, that is, $\Sigma = \{0, 1\}$ unless otherwise specified. We denote M and N to be Turing machines or finite state automata, L to be a language, and A and B to be sets. The parallel bars $|\cdot|$ represents the length of a string or the cardinality of a set. Let A be a set. The complement of A is written as $\overline{A}$ where $\overline{A} = \Sigma^* - A$. The power set of A, denote $\mathcal{P}(A)$, is the set of all subsets of A. Assume that $A = \{a, b\}$. The power set of A is $\mathcal{P}(A) = \{\{\}, \{a\}, \{b\}, \{a, b\}\}$. We write $\alpha \in A$ if α is a member or an element of A; otherwise, we write $\alpha \notin A$. Let B be a set. We write $B \subseteq A$ if B is a subset of A and possibly equals to A, and $B \subsetneq A$ or $B \subset A$ if B is a proper subset of A, or B is strictly included in A.

If $\alpha \in \mathbb{C}$, we denote α^* as its complex conjugate; that is, if $\alpha = a + bi$, then $\alpha^* = a - bi$. Notice that only the image part of complex number number is flipped. From now on, we will define everything over the complex field unless otherwise specified. The same properties

and definitions apply to the real field without taking conjugation. Let x, y be two strings. Then $\langle ... \rangle$ denotes the *pair function*, which concatenates two strings, and log is base 2 unless otherwise specified.

1.1.2 Functions and Relations

A *tuple* is a finite sequence. A k-tuple is a sequence of k elements. A function that takes k arguments is called *k-ary function*. A *predicate* is a function that returns only true or false. A *k-ary relation* is a predicate whose domain is a set of k-tuples $A \times \cdots \times A$. If $k = 2$, we have a *binary relation*. An *equivalence relation* R is a binary relation satisfying the following conditions:

1. R is *reflexive* if $\forall x\ [xRx]$.
2. R is *symmetric* if $\forall x, y\ [xRy \iff yRx]$, and
3. R is *transitive* if $\forall x, y, z\ [xRy \wedge yRz \Longrightarrow xRz]$.

Definition 1.1.1 (equivalence class) Let A be a set and R be an equivalence relation. For each element $a \in A$, the *equivalence class of a*, denoted $[a]$, is the set of all elements $x \in A$ such that x is related to a by R. It is equivalently stated as

$$[a] = \{x \in A \mid xRa\}.$$

Myhill and Nerode [36] provided us the following definition to partition sets.

Definition 1.1.2 Let L be a language over alphabet Σ, the *Myhill-Nerode equivalence relation* R_L on Σ^* is defined by:

$$xR_Ly \overset{def}{\iff} \forall z \in \Sigma^*\ [xz \in L \iff yz \in L].$$

Definition 1.1.3 Let f, g be two functions map $\mathbb{N} \to \mathbb{R}^+$.

1. $f(n) = O(g(n))$ if for some $c > 0$, there exists an n_0 such that for every integer $n \geq n_0$, $f(n) \leq c \cdot g(n)$. We say that $g(n)$ is an upper bound for $f(n)$.
2. $f(n) = o(g(n))$ if for some $c > 0$, there exists an n_0 such that for every integer $n \geq n_0$, $f(n) < c \cdot g(n)$. Moreover, $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$.
3. $f(n) = \Omega(g(n))$ if for some $c > 0$, there exists an n_0 such that for every integer $n \geq n_0$, $f(n) \geq c \cdot g(n)$. We say that $g(n)$ is a lower bound for $f(n)$.
4. $f(n) = \omega(g(n))$ if for some $c > 0$, there exists an n_0 such that for every integer $n \geq n_0$, $f(n) > c \cdot g(n)$. Moreover, $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.
5. $f(n) = \Theta(g(n))$ if $O(g(n)) = \Omega(g(n))$, or equivalently $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 1$.

1.1.3 Inner-product

Let V be a vector space over $\mathbb{C}$. An inner-product, written $(.)$, over V is a complex function: $V \times V \to \mathbb{C}$ that satisfies the the following conditions:

1. *(Positivity)* $\forall v \in V\ [(v, v) \geq 0 \wedge ((v, v) = 0 \iff v = 0)]$.
2. *(Linearity)* $\forall v, u, z \in V\ [(\alpha v + \beta u, z) = \alpha(v, z) + \beta(u, z)]$, and
3. *(Skew symmetry)* $\forall v, u \in V\ [(v, u) = (u, v)]$.

Here is one of useful properties of inner-product: $(v, \alpha u) = (\alpha u, v)^* = \alpha^*(u, v)^* = \alpha^*(v, u)$ where $(u, v)^* = (v, u)$. An *inner-product space* is a vector space V over $\mathbb{C}$ together with an inner product $(...)$. An infinite dimensional inner-product space $\mathcal{H}$ over $\mathbb{C}$ is a *Hilbert space*, and it is a complete matrix space with respect to the matrix generated by inner-product. Let V be any inner-product space and v be any vector in V. We define a *linear function*, $v^* : V \to \mathbb{C}$ such that $v^*(u) = (v, u)$. The set of such linear functions is also an inner-product space, and it is known as *vector dual* of V and denoted V^*. The *norm* of a vector is determined as $\|v\| = \sqrt{(v, v)}$.

1.1.4 Linear Transformation

In any inner-product space V, there is a usual vector space basis $\{x_i\}_{i \in I}$. We define every vector $v \in V$ as a finite linear combination of basis vectors, and the basis is *orthonormal* if $\|x_i\| = 1$ and $(x_i, x_j) = 0$ for all $i \neq j$. We write each vector $v \in V$ with respect to orthonormal basis as $v = \sum_{i \in I} \alpha_i x_i$ where $\alpha_i = (x_i, v)$. Similarly, each dual vector $v^* \in V^*$ can be written as $v^* = \sum_{i \in I} \beta_i x_i^*$ where $\beta_i = (v, x_i)$. An operator is a map that transforms one vector into another, and it can be expressed as: $A : v \to u$. Each element $v \in V$ can be visualized as a column vector of the α_i's, and each element $v^* \in V^*$ can be visualized as a row vector of the β_i's. A *linear operator* U is a square matrix with row and column both indexed by I that is represented by a set of *matrix elements* $\{(x_i, Ux_j)\}_{i,j \in I}$. A linear operator must satisfy the following conditions:

1. $U(v + u) = Uv + Uu$, and
2. $U(\alpha u) = \alpha U u$

where α is a complex constant, and v and u are vectors. If U is described as $\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$, the operator U applied to the vector v (written $Uv = u$) where $v = ax + by$ and $u = a'x + b'y$ are in the standard basis $\{x, y\}$, can be expressed as $\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} a' \\ b' \end{pmatrix}$. U^* is called the *adjoint* of U if $(Uv, u) = (v, U^*u)$. If an operator is its own adjoint: that is, $U^* = U$, it is called *Hermitian* or *self-adjoint*. The linear operator is called *unitary* if its adjoint exists, and it satisfies $U^*U = UU^* = I$ where I is an identity operator. In addition, a matrix is *stochastic* if each of its rows has nonnegative real entries that sum up to 1.

Lemma 1.1.4 *Let V be an inner-product space and U be a linear operator on V. Assume that U^* exists. Then U preserves norm if and only if $U^*U = I$.*

Proof. Note that for any $v \in V$, $\|Uv\|^2 = (Uv, Uv) = (v, U^*Uv)$. It is clear that if $U^*U = I$, then $\|Uv\|^2 = (v, v) = \|v\|^2$. Thus U preserves norm. For the converse, let $M = U^*U - I$. Since U preserves norm, $(v, U^*Uv) = \|Uv\|^2 = \|v\|^2 = (v, v)$ for any $v \in V$. Therefore, for every $v \in V$, $(v, Mv) = (v, (U^*U - I)v) = (v, U^*Uv) - (v, v) = 0$. Thus $M = 0$, and it follows that $U^*U = I$. □

An *eigenvalue* $\lambda \in \mathbb{C}$ of an operator U is a scalar such that for some nonzero vector v, $Uv = \lambda v$. Every vector that satisfies this relation is called an *eigenvector* of operator U belonging to the eigenvalue λ. Moreover, if λ is positive, v does not change direction when it is operated by U; if λ is negative, the direction of v reverses when it is operated by U; and if λ is zero, v is transformed into the zero vector by U.

Let v be a nonzero vector such that $U(x) \cdot v = \lambda \cdot v$ where λ is an eigenvalue of $U(x)$. Then the determinant of $[U(x) - \lambda \cdot I^{n}]$ has the following form:

$$\begin{aligned}
\det|U(x) - \lambda \cdot I^{n}| &= \det\left|\begin{pmatrix} \gamma_{1,1} & \cdots & \gamma_{1,n} \\ \vdots & \ddots & \vdots \\ \gamma_{n,1} & \cdots & \gamma_{n,n} \end{pmatrix} - \begin{pmatrix} \lambda & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda \end{pmatrix}\right| \\
&= \det\left|\begin{pmatrix} \gamma_{1,1} - \lambda & \cdots & \gamma_{1,n} \\ \vdots & \ddots & \vdots \\ \gamma_{n,1} & \cdots & a_{n,n} - \lambda \end{pmatrix}\right| \\
&= [(\gamma_{1,1} - \lambda) \cdots (\gamma_{n,n}) - \lambda] + \cdots + [(\gamma_{1,n}) \cdots (\gamma_{n,n-1})] \\
&\quad -[(\gamma_{n,1}) \cdots (\gamma_{1,n})] + \cdots + [(\gamma_{1,1} - \lambda) \cdots (\gamma_{2,n})]
\end{aligned}$$

The *characteristic polynomial* can be written as follows:

$$\begin{aligned}
0 &= 1 \cdot \lambda^n - a_{n-1}\lambda^{n-1} - \cdots - a_1\lambda - a_0, \text{ or} \\
1 \cdot \lambda^n &= a_{n-1}\lambda^{n-1} + \cdots + a_1\lambda + a_0
\end{aligned}$$

where each a_k is composed of $\gamma_{i,j}$ with $1 \le i, j \le n$.

1.2 Formal Language

In 1959 Chomski [12] introduced the hierarchy of grammars (see Figure 1.3), known as Chomsky hierarchy, which comprises four types of languages: regular language, context-free language, context-sensitive language, and recursive enumerable language. He also showed that the class of languages recognized by Turing machines is recursive enumerable.

1.2.1 Regular Languages

Finite State Automata

Finite state automata were first studied by Huffman [37] (1925-1999). Mealy [55], and Moore [58] in the mid 1950's. There was no concept of nondeterminism until 1959, when Rabin and Scott [66] introduced the idea of nondeterministic machines along with their decision problem. Later, in 1963 Rabin [65] generalized their finite automata and introduced probabilistic automata.

Intuitively, a deterministic finite state automaton, shown in Fig 1.1, is the simplest computing device that consists of a set of states, an input alphabet, a set of rules for moving, an initial state, and a set of final states. This device can only read symbols and according to some rules, change from one state to another while moving the head from one symbol to the next one within the input range.

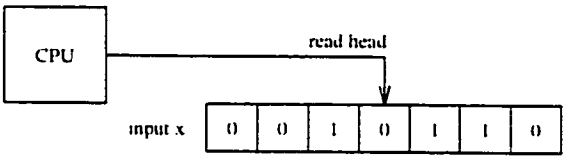

Figure 1.1: The schematic of a finite state automaton.

Definition 1.2.1 (Deterministic Finite State Automata) A deterministic finite state automaton (or DFA for short) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

Q is a finite set of internal states,

Σ is a finite alphabet with a distinguished blank symbol $\{\sqcup\} \notin \Sigma$,

δ is a transition function: $Q \times \Gamma \longrightarrow Q$,

$q_0 \in Q$ is the initial state, and

$F \subseteq Q$ is the set of final states.

Definition 1.2.2 (recognizability) Let L be a language over alphabet Σ and M be a Turing machine. We say that M *recognizes* L if for every input x, $x \in L$ if and only if M accepts x. If L is recognized by some Turing machine, L is *recursively enumerable (or r.e. for short)*. In addition, the complement of L is *co-r.e.*

Definition 1.2.3 A language L is regular if and only if some DFA recognizes it.

Definition 1.2.4 The class of regular languages is denoted as **REG**.

Example 1.2.5 The following DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizes $L = \{w \mid w$ starts with 0 and had odd length, or starts with 1 and has even length $\}$.

1. $Q = \{q_0, q_1, q_2, q_3, q_4\}$.
2. $\Sigma = \{0, 1\}$.
3. $F = \{q_1, q_4\}$. and
4. δ can be represented as the following transition table

	0	1
q_0	q_1	q_3
q_1	q_2	q_2
q_2	q_1	q_1
q_3	q_4	q_4
q_4	q_3	q_3

or as the following finite state diagram.

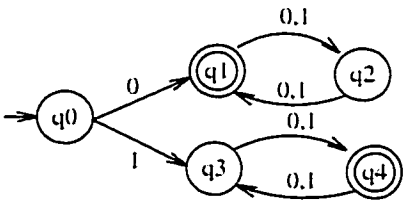

Figure 1.2: A finite state diagram.

Lemma 1.2.6 **REG** *is closed under complementation.*

Proof. Let L be any language over Σ. Then $\overline{L} = \Sigma^* - L = \{x | x \in \Sigma^* \text{ and } x \notin L\}$. Assume that a DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizes L. A new DFA $M' = (Q', \Sigma', \delta', q_0', F')$ is defined as follows:

1. $Q' = Q$.
2. $\Sigma' = \Sigma$.
3. $\delta' = \delta$.
4. $q_0' = q_0$. and
5. $F' = Q - F$.

We want to show that M' recognizes $\overline{L}$. We need to show that for every x, if $x \in \overline{L}$, M' accepts x; and if M' accepts x, $x \in \overline{L}$.

Case1: Since $\delta' = \delta$, any computation of M on input x is the computation of M' on the same input. Assume that $x \in \overline{L}$. Clearly, M does not accept x since $x \notin L$. Thus when M halts, it is in a state not in F. From the definition of F', it follows that M' accepts x. Therefore, if $x \in \overline{L}$, M' accepts x.

Case2: Assume that M' accepts x. Namely, M' halts in a final state in F'. This implies that M does not accept x since $F' \cap F = \emptyset$. Thus x is in $\Sigma^* - L = \overline{L}$.

□

To make use of Definition 1.1.2, Condon [15] provided a nice lemma that builds a bridge between Myhill-Nerode equivalence relation and regular languages.

Lemma 1.2.7 ([15]) *Let L be a language over alphabet Σ. Suppose that there exists an equivalence relation E on Σ^* such that*

1. *the number of equivalence classes of E is finite, and*
2. *for all x and y, xEy implies xR_Ly.*

Then L is regular.

The pumping lemma for regular languages was first introduced in 1961 by Bar-Hillel, Perles, and Shamir [4]. It is used to show the non-regularity of a language.

Lemma 1.2.8 (pumping lemma for regular languages [68]) *Let L be a language. If L is regular, there exists a number p (called pumping length) such that for any string $w \in L$ of length at least p, w can be divided into $w = xyz$ satisfying the following conditions:*

1. *for each $i \geq 0$, $xy^iz \in L$,*
2. *$|y| > 0$, and*
3. *$|xy| \leq p$.*

The formal proof is given in Appendix A.

1.2.2 Context-Free Language

Definition 1.2.9 (context-free grammar) A context-free grammar is a 4-tuple (V, Σ, R, S) where

1. V is the set of variables,
2. Σ is the set of terminals that is disjoint from V,
3. R is the set of rules, each consists of a variable, a string of variables, and terminals; and

4. S is the initial variable.

Example 1.2.10 To show that $L = \{a^m b^n c^n \mid m, n \geq 0\}$ is context-free, we define the following CFG. Let $G = (V, \Sigma, R, S)$ be a CFG for L where

1. $V = \{X, S, T\}$.
2. $\Sigma = \{a, b, c\}$, and
3. R is defined :

$$S \longrightarrow TX.$$
$$T \longrightarrow Ta \mid \epsilon, \text{ and}$$
$$X \longrightarrow bXc \mid \epsilon \text{ where "|" is used as an "or".}$$

Definition 1.2.11 A language is called *context-free* if there is a context-free grammar that generates it.

Definition 1.2.12 **CFL** is the collection of context-free languages.

Lemma 1.2.13 (pumping lemma for context-free languages [68]) *Let L be a language. If L is context-free, there exists a number p (called pumping length) such that for any string $w \in L$ of length at least p, w can be divided into $w = uvxyz$ satisfying the following conditions:*

1. *for each $i \geq 0$, $uv^i xy^i z \in L$,*
2. *$|vy| > 0$, and*
3. *$|vxy| \leq p$.*

The formal proof is given in Appendix A.

1.2.3 Turing Machines

In general, a Turing machine is a computing device that consists of an infinitely bilateral tape, a read/write head, and a finite state control.

We allow the tape to be both readable and writable and possibly divided into multiple tracks. The tape consists of infinitely many cells indexed by $\mathbb{Z}$ to both ends. The tape head is initially at the start cell with index 0. At each step, the tape head is allowed to move along the tape, one cell away, either left or right to its adjacent cell, from its current location or stays still. In the classical Turing machine model, each cell contains classical bits 0 or 1 where in quantum case, each cell contains a quantum bit. The finite control changes from state(s) to state(s) according to some rules defined by a transition function.

A *segment* is a region on the tape that consists of only finitely many non-blank symbols. A *configuration* of a Turing machine is the full descriptions of the tape content, the head position,

and the current state of the finite control. At all times, only a finite number of tape cells may contain non-blank symbols. For each configuration c of some Turing machine M, we denote the successor configuration c' as a result from one application of the transition function to the current state q and the current symbol σ. The notation $c \rightarrow_M c'$ means that configuration c *precedes* configuration c'. For convenience, we define the initial configuration of M as when the tape head is at its *start cell*, and the machine is at state q_0. An *initial configuration* has an input x that consists of non-blank symbols for finitely numbered cells, and all other tape cells are blank. We say that M *halts on input* x if it eventually enters its final state q_f. The *running time* of a Turing machine is the minimal number of steps that M halts on input x. For convention, we write $M(x)$ if M accepts x and $\overline{M}(x)$ if M rejects x. Moreover, $|M(x)|$ denotes the number of accepting paths and vice versa.

Let $T(n)$ denote a time-bounding function of a given Turing machine, which maps from $\mathbb{N}$ to $\mathbb{N}$. The running time of M on x is defined in accordance to each variant of Turing machines as some function $T(n)$ with respect to the length of input x. In general, the output of M is a character in Σ returned by M when halting occurs. If M calculates a function, its output is a string in Σ^*, which consists of those tape contents from the leftmost non-blank symbol to the rightmost non-blank symbol or the empty string if the entire tape is blank. Moreover, a machine is said to accept or reject the given input if, at the end of computation, it returns 1 or 0, respectively; otherwise, the outcome is undetermined.

In addition, we give the formal definition for each variant of Turing machine at the beginning of each related section. However, in this thesis, we deal only with higher level description of Turing machines. It is not difficult to show the equivalence of the two.

1.2.4 Universal Turing Machines

The idea of universal Turing machines is to construct a machine that reads an encoding of another machine with its input, then interprets and simulates its behaviors. Imagine a Turing machine as a program and a universal Turing machine as a computer. We can surely run any program of correct format using a computer. A real-life example is to run a Java program on a Java platform or to compile a Java program using Java interpretor. Let U be a universal Turing machine that takes the encoded description of M concatenated with the input x of M. The input w of the machine U is the binary encoding of the form $\langle M, x \rangle$. We write $U(\langle M, x \rangle) = M(x)$ to denote the simulation of a universal Turing machine U over the behaviors of an arbitrary M on its input x.

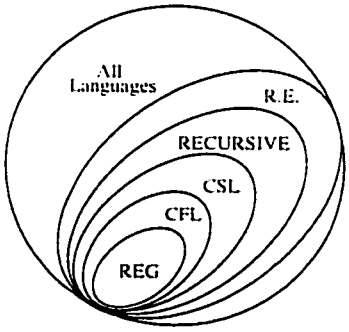

Figure 1.3: The Chomsky hierarchy.

Chapter 2

Deterministic and Nondeterministic Classes

2.1 Deterministic Computations

2.1.1 Background

In 1963 Evey [26] showed the equivalence of nondeterministic and deterministic Turing machines without time bound. Since we often want to compare the efficiency of algorithms, imposing time and space bounds becomes essential.

It is shown that the simulation of some two-tape deterministic Turing machine by any one-tape deterministic Turing machine requires quadratic slowdown. This implies that, in polynomial time, the number of tapes does not make any difference in terms of computability. This gives researchers some convenience to use multi-tape machines without effecting the running time. However, things are different with the one-tape linear-time restriction. An obvious example is to make a copy of an input string right next to it (if we copy it right below it on the second track, it takes only real time.). Since, each time, the head can carry a one-bit information, and it must repeat for at least n times where n is the length of input, any one-tape Turing machine requires at least $O(n^2)$ time.

How much computation power does a one-tape linear-time deterministic Turing machine possess? Surprisingly, it has the same computation power as finite state automata [33, 44]. In linear time, the ability of changing the contents of tape cells and the flexibility of using at most linear-space[1] do not give any advantage in computation.

2.1.2 Deterministic Turing Machines

Figure 2.1 illustrates the behaviors of one-tape deterministic Turing machines. The tape head shifts, reads and/or writes according to some transition function. A symbol σ is overwritten

[1]Note that if we impose a time bound, we, at the same time, impose a space bound. However, the backward implication does not apply.

by τ along the computation. At finite state control (or CPU), the current state changes accordingly.

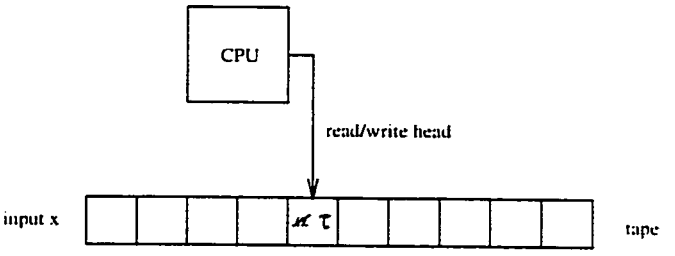

Figure 2.1: A schematic for one-tape deterministic Turing machines.

Formally, the definition of a one-tape deterministic Turing machine is given as follows.

Definition 2.1.1 (one-tape deterministic Turing machines) A one-tape deterministic Turing machine (or 1DTM for short) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

Q is a finite set of internal states.

Σ is a finite alphabet with a distinguished blank symbol $\{\sqcup\} \notin \Sigma$.

Γ is a tape alphabet with $\Sigma \cup \{\sqcup\} \subseteq \Gamma$.

δ is a transition function: $(Q - \{q_{acc}, q_{rej}\}) \times \Gamma \longrightarrow Q \times \Gamma \times \{L, N, R\}$.

$q_0 \in Q$ is the initial state.

$q_{acc} \in Q$ is the accepting state, and

$q_{rej} \in Q$ is the rejecting state where $q_{rej} \neq q_{acc}$.

Each entry of the transition function is described as $\delta(q, \sigma) = (q', \tau, d)$ where q is the current state, q' is the next state, $\sigma \in \Gamma$ is the symbol read at the current cell, $\tau \in \Gamma$ is the symbol written over the same cell, and $d \in \{L, N, R\} = \{-1, 0, 1\}$ is the direction of the head move. Directions L and R represents the head movement to left and right, respectively, and direction N represents no movement.

Definition 2.1.2 ($T(n)$-time 1DTM) Let $T: \mathbb{N} \to \mathbb{N}$. A $T(n)$-time 1DTM (abbreviated $T(n)$-1DTM) is a one-tape deterministic Turing machine M such that, for any input x, the number of steps of M on input x is at most $T(|x|)$.

Definition 2.1.3 Let $T: \mathbb{N} \to \mathbb{N}$. We say that a DTM M is $T(n)$ *time-bounded* if it always halts in either q_{acc} or q_{rej} at most $T(n)$ steps where n is the size of input. The class **1-DTime**$(T(n))$ is defined as the set of languages L such that there exists a $T(n)$-1DTM that recognizes L.

2.1.3 Crossing Sequence

In 1965 Hennie [33] used the *crossing sequence technique* to show that, in linear time, any one-tape deterministic Turing machine is equivalent to some deterministic finite state automaton. The idea of the crossing sequence argument is that, instead of looking at the whole computation of a Turing machine, one simply observes the series of ordered states crossed over some inter-cell boundary (a boundary between two adjacent cells). Upon being given those sequences, we can retrieve the whole computation. Later, in 1985 Kobayashi [44] improved the result and showed that any one-tape $o(n \log n)$-time deterministic Turing machine is also equivalent to some finite state automaton. The deterministic crossing sequence technique is given as follows.

Definition 2.1.4 (crossing sequence for deterministic computation [33]) Let M be a $T(n)$-1DTM. Each pair of adjacent cells on the tape of M is separated by an inter-cell boundary (a boundary between two adjacent cells). In the computation of M, consider an inter-cell boundary b and the sequence of states of M at the steps when the head crosses b, first from left to right then alternatingly in both directions. This ordered sequence of states is called the *crossing sequence* at the inter-cell boundary b in the computation of M.

Definition 2.1.5 Any inter-cell boundary between the right-boundary and the left-boundary of x (including both ends) is called a *critical-boundary* of x.

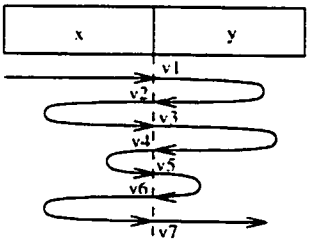

Figure 2.2: An example for the deterministic crossing sequence technique.

The following two properties are keys to the crossing sequence technique. They are based on the fact that a machine cannot distinguish two segments of computations if they share the same crossing sequence. Hence we can replace or cut a segment of computation.

Lemma 2.1.6 (replace) *Let M be a Turing machine, and x_1y_1 and x_2y_2 be two input strings. Suppose that b_1 and b_2 are the inter-cell boundaries for x_1y_1 and x_2y_2, respectively. If b_1 and b_2 are identical, the strings x_1y_1, x_1y_2, x_2y_1, and x_2y_2 are treated identically. In other words, if M accepts x_1y_1 and x_2y_2, M accepts x_1y_2 and x_2y_1 and vice versa (see Figure 2.3).*

Proof Sketch. Upon given a sequence of states, say s, at the boundary i on input x, since the head always starts at the leftmost symbol of x, every odd-th state in s is a crossing from left to right at i, and every even-th state is a crossing from right to left. Without loss of generality, we assume that if the tape head stops on the right of the boundary i, it accepts:

otherwise, it rejects. This can be done simply by moving the head to the desired side of i at the end of the computation without effecting the total running time.

Now suppose that we divide the computation of x into two segments, say t_L and t_R, which cover only non-blank cells. Notice that we, in fact, divide the sequence s such that at the boundary i, t_L contains all the even-th states and t_R contains all the odd-the states. The idea is that, whenever the head moves out of the boundary, we push it back and change it to the next odd-th/even-th state, respectively according to the sequence s. Hence we separate the computations on both sides and we can reconstruct them by matching the corresponding sequences. Therefore, if b_1 and b_2 are identical, x_1y_1, x_1y_2 x_2y_1, and x_2y_2 cannot be distinguished. For the formal definition, see Definition 3.1.6. ■

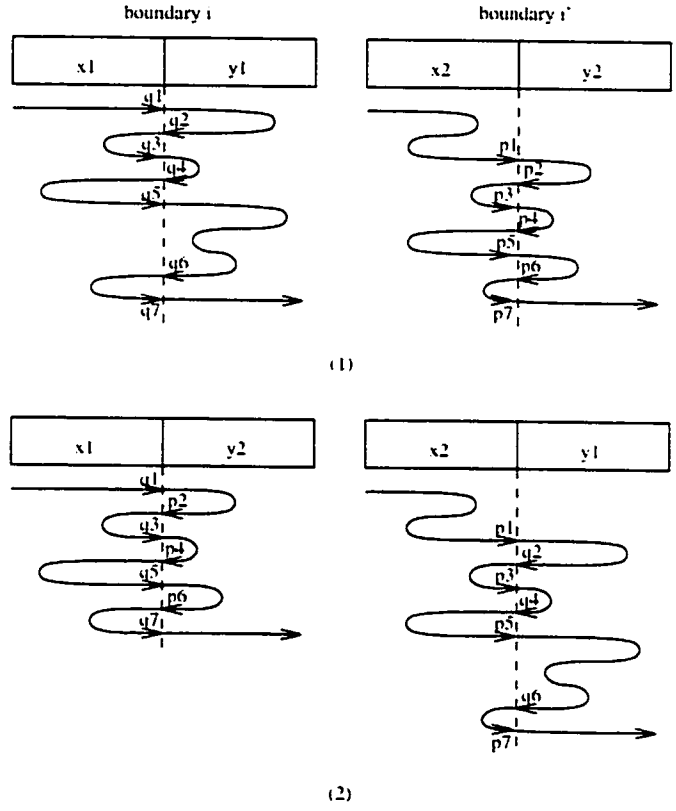

Figure 2.3: An example of the replace operation.

Lemma 2.1.7 (cut) *Let xyz be an input string and b_1 and b_2 be the inter-cell boundaries for xy and yz, respectively. If b_1 and b_2 are identical, then xyz and xz are treated identically (see Figure 2.4).*

Proof. Let t_1, t_2, t_3, t_4 be the tape segments for x, yz, xy, and z, respectively. According to lemma 2.1.6, t_1t_4 and t_3t_2 must be treated identically. Since $t_1t_4 = xz$ and $t_1t_2 = xyz$, xyz and xz are treated identically. □

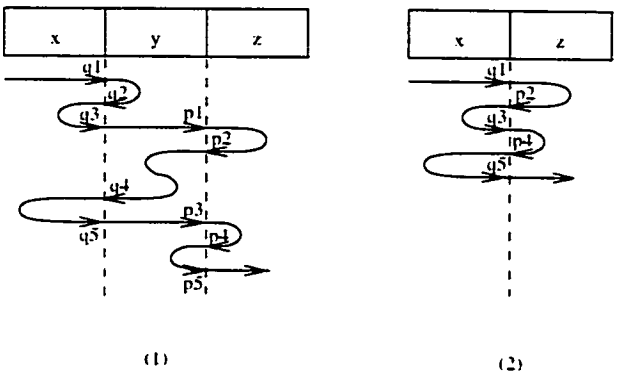

Figure 2.4: An example of the cut operation.

2.1.4 Kolmogorov Complexity

In 1965 Kolmogorov proposed his idea about measuring the amount of mutual information in one object about another. It became apparent that some objects are more complex than others. Thereafter, numerous researchers have continued developing the idea of Kolmogorov complexity. Kolmogorov complexity is a complexity measure based on the existence of universal Turing machines. According to Kolmogorov, a complexity can be measured by the length of the shortest program run by a universal Turing machine to produce the desired data correctly. It has been shown that the corresponding complexities to different choices of universal Turing machines only differ by at most an additive constant.

Information is one of the fundamental concepts in computer science. Intuitively, we want to measure the amount of information a given string contains. In general, the more the string "looks random" the more the information it contains. In other words, we are looking at the structure (or randomness) of a string. For example, upon given strings $x = 00000000$ and $y = 01001011$, we can say that x is "easier to construct" than y by observing their patterns. To construct x, we can possibly give an instruction: "write eight zeros" where in comparison, specifying the value of each bit might be necessary to construct y. Note that we can also construct x by specifying each bit. Obviously, the length of the description for constructing a given object depends crucially on the specification method. However, we want the descriptional complexities of x and y to depend only on the strings themselves. Therefore, we restrict our specification methods to be effective and universal. We then are able to compare the descriptional complexities of x and y independently and say that y is more complex than x. This concept of complexity is known as *Kolmogorov complexity*.

Resource Unbounded Kolmogorov Complexity

In retrospect, the Kolmogorov complexity of a string is the shortest program that produces it. Formally, we describe the Kolmogorov complexity relatively to some Turing machine.

Definition 2.1.8 Let M be a Turing machine. Then for any $x \in \Sigma^*$, the Kolmogorov complexity of x, denote $K(x)$, with respect to M is defined as

$$K_M(x) = min\{|p| \mid M(p) = x\}.$$

The next proposition says that the Kolmogorov complexity of any string x is at most the sum of the length of x itself plus the length of the description of some machine, which is a constant.

Proposition 2.1.9 *There exists a constant c such that for any input x, $K(x) \leq |x| + c$.*

Proof.

Let M be a Turing machine that halts right after it starts. Hence M reads the input x and outputs x. The description of x is the encoding of M concatenated with x: that is, $\langle\langle M \rangle, x\rangle$. Let $c = |\langle M \rangle|$, and the proof is completed.

□

We previously mentioned that the complexity measure should purely depend on the input. Hence we want to show that the choice of universal Turing machines is irrelevant.

Proposition 2.1.10 *Let U_1 and U_2 be two universal Turing machines. Then there exists a constant c such that for any input x, $K_{U_1}(x) \leq K_{U_2}(x) + c$ where c depends only on U_1 and U_2.*

Proof. Since a universal Turing machine can simulate any Turing machine including another universal Turing machine, we can assume that U_1 simulates U_2 on w where $w = \langle U_2, x\rangle$. By letting $c = |\langle U_1 \rangle| + |\langle U_2 \rangle|$, we complete the proof. □

Hence we assume that the universal Turing machine is fixed without any loss of generality.

Incompressibility

Obviously, there exist strings that can be described by some significantly shorter programs. These strings are *compressible*. *Incompressible* strings, on the other hand, are pattern-less. Intuitively, a compressible string has a description of length c bits shorter than its length where c is some constant. On the other hand, the length of any incompressible string is its own length.

Definition 2.1.11 A string x is said to be *c-compressible* if $K(x) \leq |x| - c$. If x is not c-compressible, it is incompressible.

The next theorem shows the existence of incompressible strings.

Theorem 2.1.12 *For each length, there exist incompressible strings.*

Proof. Let Σ^n be the set of strings of length n. The number of binary strings x of length n is 2^n. The number of descriptions $d(x)$ shorter than n is $\Sigma_{i=0}^{n-1} 2^i = 2^0+2^1+2^2+\cdots+2^{n-1} = 2^n-1$. Since each description describes at most one string, and the number of the descriptions is less than the number of strings of length n, there exists at least one string of length n that is incompressible. □

Corollary 2.1.13 *For each length n and each constant $c \in \mathbb{N}^+$, there are at least $2^n - 2^{n-c+1} + 1$ strings x with $|x| \leq n$ that are incompressible by c.*

Proof. We apply an argument similar to the proof of Theorem 2.1.12. The number of binary strings x of length n is 2^n. The number of descriptions $d(x)$ shorter than $n-c$ is $2^{n-c+1}-1$. Equivalently, there are at most $2^{n-c+1}-1$ strings of length n that are c-compressible (or have Kolmogorov complexity $K(x) \leq n-c$). Hence there exist at least $2^n - 2^{n-c+1} + 1$ strings that are incompressible by c. □

The following lemma was stated in [50]. We give a different proof that is easier to understand.

Lemma 2.1.14 *Let x^R be the reverse of x. Then the set of palindromes L_{pal} is defined to be $\{xy \mid x = y^R \wedge x, y \in \Sigma^*\}$. Let M be any 1DTM. Then M recognizes L_{pal} in $\Omega(n^2)$ steps.*

Proof. In order to apply Kolmogorov complexity for this proof, we need to use the crossing sequence technique. In general, we want to count the number of the "crossings" on each boundary since the running time is equal to the summation of the number of states listed in all boundaries crossed.

By Corollary 2.1.13, assume that the given string x of length n has Kolmogorov complexity $K_M(x) \geq n$. The essential idea is to find the minimum information required to construct x. M takes an input w of the form $x0^n y$ and runs in time $T(n)$. Let $S(n,i)$ be the crossing sequence at the position i of the segment 0^n. To indicate the inter-cell boundary at the position i, we write the input in the form $x0^i0^{n-i}y$. We first consider the case where the length of every crossing sequence at each boundary is greater than the average running time spent on each boundary. Assume that $|S(n,i)| > \frac{T(n)}{n}$ for all i. Then we have

$$T(n) = n \cdot \frac{T(n)}{n} < \sum_{i=1}^{n} |S(n,i)|.$$

Since the total number of "crossings" during the computation is less or equal to the total running time, we have

$$T(n) < \sum_{i=1}^{n} |S(n,i)| \leq T(n).$$

Therefore, we have a contradiction since $T(n) < T(n)$ is obtained. This implies that there exists some compressible information at some boundary. We now assume that there exists an i such that $|S(n,i)| \leq \frac{T(n)}{n}$. We first prove the following claim.

Claim 1 *If $S(n,i) = S(n',i')$, then $n = n'$ and $i = i'$ for all n, n', i and i'.*

Proof of Claim 1. Consider two strings $x0^i0^{n-i}y, x'0^{i'}0^{n'-i'}y' \in L_{pal}$ where $S(n,i)$ is the crossing sequence at i on the first string, and $S(n',i')$ is the crossing sequence at i' on the second string. By Lemma 2.1.6 and assumption that $S(n,i) = S(n',i')$, we have $x0^i0^{n'-i'}y', x'0^{i'}0^{n-i}y \in L_{pal}$. If $x0^i0^{n'-i'}y' \in L_{pal}$ (the other case is symmetric), we have $x = x'$, $y = y'$ and $x = y^R = y'^R$; hence $n' = n$. Since $i + n' - i' = n$, we have $i = i'$. □

Claim 1 implies that, by using the crossing sequence technique, we can reconstruct x by verifying all binary strings of length n. In other word, we rebuild the string x by checking each candidate of binary strings of length $|x|$. Hence we have

$$\begin{aligned} n = |x| &\leq K_M(x) \qquad \text{(by assumption)} \\ &\leq |S(n,i)| + \log i \end{aligned}$$

where $\log i$ is the location of i in binary representation. Since $\log i \leq \log n$, we have

$$n \leq |S(n,i)| + \log n.$$

Recall that by assumption, $|S(n,i)| \leq \frac{T(n)}{n}$, we have $n \leq \frac{T(n)}{n} + \log n$. By simplification, we have

$$\begin{aligned} T(n) &\geq n \cdot (n - \log n) \\ &= n^2 - \frac{\log n}{n} \qquad (\tfrac{\log n}{n} \text{ is negligible}) \\ &\geq c \cdot n^2. \end{aligned}$$

□

The next proposition follows essentially the proof of Lemma 2.1.14.

Proposition 2.1.15 *Let M be any 1DTM. Then M recognizes $\overline{L_{pal}}$ in $\Omega(n^2)$ steps.*

Similarly, we prove the following lemma.

Lemma 2.1.16 *Let M be any 1DTM. Then M recognizes L_{pal} in $\Theta(n^2)$ steps.*

Proof. The lower bound is obtained by Lemma 2.1.14. To show the upper bound, we construct a 1DTM M that recognizes L_{pal} in $O(n^2)$ time. This can be done by checking each pair of corresponding bits on x and y back and forth. Since there are n pairs to check and the of input is n; thus the running time is quadratic. Therefore, $T(n) = \Omega(n^2) = O(n^2) = \Theta(n^2)$. □

2.2 Nondeterminism

2.2.1 Background

Since the idea of nondeterminism had been presented [66], it became a field that has inspired immmerably many researchers. Nevertheless, there are important properties that are still unknown. For example, it has been well-known that **NP** is closed under union and intersection; however, the complementation property has remained open for sometime. In this section, we give a partial solution to this problem. Under the one-tape $o(n \log n)$-time restriction, we prove the equivalence of nondeterministic computations and co-nondeterministic computations; that is, the complementation property holds. Moreover, they both collapse to **REG**. Such a result implies that neither nondeterminism nor co-nondeterminism gives any advantage in computation with such limitation. In what follows, we relax the time bound a little to be $O(n \log n)$. This relaxation has given both nondeterminism and co-nondeterminism advantages over some deterministic computation.

2.2.2 Nondeterministic Turing Machines

Intuitively, a nondeterministic Turing machine is a deterministic Turing machine that has the power of making "nondeterministic choice". The concept of nondeterminism is to allow a machine to perform different configurations in the following step. In Figure 2.5, two nondeterministic choices are made at each step, and according to each choice, the machine performs a deterministic computation.

Similar to deterministic Turing machine, we allow our nondeterministic Turing machines to be equipped with an infinitely bilateral tape, a read/write head and a finite state control. Formally, a one-tape nondeterministic Turing machine is defined as follows.

Definition 2.2.1 (one-tape nondeterministic Turing machines) A one-tape nondeterministic Turing machine (or 1NTM for short) M is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F_{acc}, F_{rej})$ where

Q is a finite set of internal states,

Σ is a finite alphabet with a distinguished blank symbol $\{\sqcup\} \notin \Sigma$,

Γ is a tape alphabet with $\Sigma \cup \{\sqcup\} \subseteq \Gamma$,

δ is a transition function: $(Q - F_{acc} - F_{rej}) \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, N, R\})$.

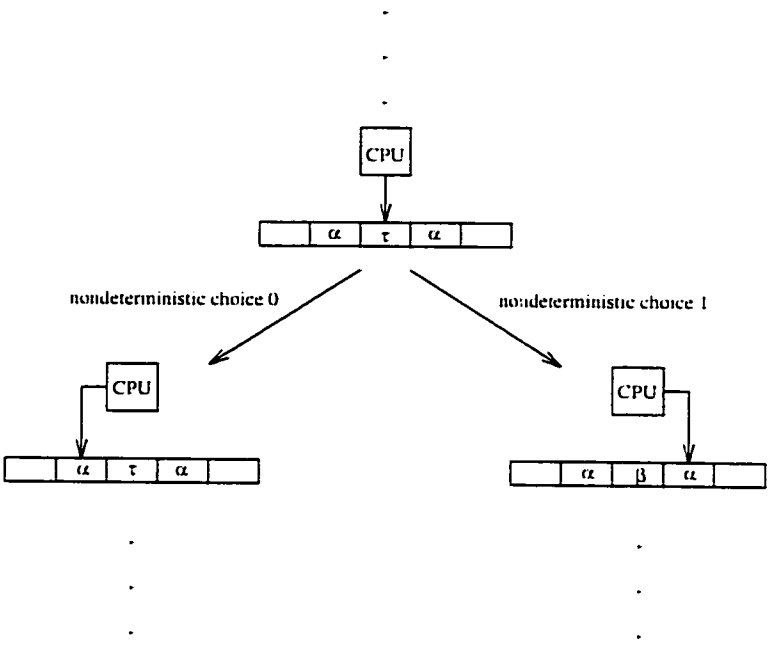

Figure 2.5: A schematic for a one-tape nondeterministic Turing machine.

q_0 is the initial state.

$F_{acc} \subset Q$ is a set of accepting states, and

$F_{rej} \subset Q$ is a set of rejecting states where $F_{acc} \cap F_{rej} = \emptyset$.

If we restrict the number of nondeterministic choice to 1, we, in fact, have a deterministic Turing machine. This leads to a trivial inclusion that $\mathbf{P} \subseteq \mathbf{NP}$. Similar to deterministic Turing machines, we define the recognizability for nondeterministic Turing machines. However, since we allow a computation to "branch" (see Figure 2.6), each path in a computation tree must be considered.

Definition 2.2.2 ($T(n)$-time 1NTM) Let $T: \mathbb{N} \to \mathbb{N}$. A one-tape $T(n)$-time 1NTM (abbreviated $T(n)$-1NTM) is a one-tape nondeterministic Turing machine M such that, for each input x, the length of every computation path of M on input x is at most $T(|x|)$.

Definition 2.2.3 Let $T: \mathbb{N} \to \mathbb{N}$. We say a $T(n)$-1NTM M recognizes a language L if for every x,

$$\begin{aligned} x \in L &\implies |M(x)| > 0, \text{ and} \\ x \notin L &\implies |M(x)| = 0. \end{aligned}$$

The class **1-NTime**($T(n)$) is defined as the set of languages L such that there exists a $T(n)$-1NTM that recognizes L.

Similarly, we define co-**1-NLIN**. Figure 2.6 depicts the computation tree for both nondeterministic computation and co-nondeterministic computation. In (a), an NTM accepts if there exists an accepting path (the results of the remaining paths do not matter), and rejects if all paths lead to the rejecting states where in (b), an NTM accepts if all computation paths accept, and rejects if there exists a rejecting path (similarly, the results of the other paths do not matter). Moreover, both nondeterministic and co-nondeterministic computations are defined using nondeterministic Turing machines. It is only the accepting criteria that makes the differences.

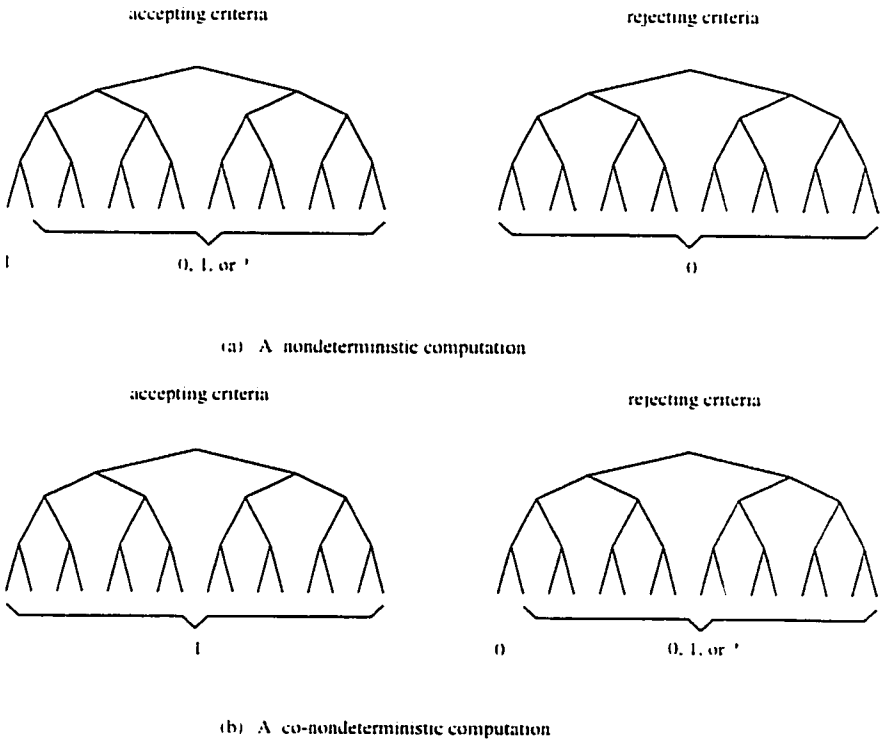

Figure 2.6: Computation trees for a nondeterministic computation and a co-nondeterministic computation.

Definition 2.2.4 (crossing sequence for nondeterministic computation [51]) Let M be a $T(n)$-1NTM. Each pair of adjacent cells on the tape of M is separated by an inter-cell boundary. In a computation path s of M, consider an inter-cell boundary b and the sequence of states of M at the steps when the head crosses b, first from left to right then alternatingly in both directions. This ordered sequence of states is called the crossing sequence at the inter-cell boundary b in the computation path s of M.

2.2.3 1-NTime($o(n \log n)$)

Michel [57] showed the existence of some non-regular set in **1-NLIN** by using a weaker definition. For our result, we generalize Definition 2.2.2 to require only accepting computation paths to have bound $T(|x|)$.

As our main result in [51], we expand Kobayashi's earlier result on one-tape $o(n \log n)$-time deterministic Turing machines and show that any language recognized by some $o(n \log n)$-time 1NTM is regular.

Definition 2.2.5 (n-fold $T(n)$-1NTM) Let $T: \mathbb{N} \to \mathbb{N}$. An n-fold $T(n)$-1NTM is a nondeterministic Turing machine M such that

1. the whole computation is folded under the segment of input, and
2. for each input x, the length of every computation path of M on input x is at most $T(|x|)$.

Intuitively, we can simulate a machine's computation in a zig-zag fashion using only the space under the input segment.

Lemma 2.2.6 *Let $T(n) = o(n \log n)$. Then n-fold $T(n)$-1NTM is equivalent to regular $T(n)$-1NTM.*

Proof Sketch. The direction that any n-fold $T(n)$-1NTM is a regular $T(n)$-1NTM is trivial. To show the second direction, let N be a regular $T(n)$-1NTM. Note that, within $T(n)$ time, N cannot not visit more than $n(\log n - 1)$ cells of blank tape. Let N' be a $T(n)$-1NTM with $o(n \log n)$ tracks that simulates N. N' first writes the input x on the first track and behaves exactly the same until the head of N moves out of the boundary, say the rightmost bit of x. Whenever that happens, N' turns around and performs the computation on the next track. Each time N' moves beyond n-th cell, N' simply turns around and simulates the computation on the next track in zig-zag fashion. At the end of simulation, N' "folds" the computation that is outside of rightmost boundary of x under the range of x. Similarly, we construct another $T(n)$-1NTM N'' that folds the computation of N' outside of the other end. Therefore, we fold the entire computation of N underneath of x. ∎

Using essentially Kobayashi's technique in [44], we prove the following lemma.

Theorem 2.2.7 ([51]) *Let M be an n-fold $o(n \log n)$-1NTM. Then there exists $c \in \mathbb{N}$ such that for every x, the length of the crossing sequence of every inter-cell boundary in every computation path of M on input x is at most c.*

Proof. Our poof extracts the crossing sequence technique used in [44]. Let $T(n)$ be the running time. Recall that

$$T(n) = o(n \log n) \iff \lim_{n \to \infty} \frac{n \log n}{T(n)} = \infty.$$

Let q be the number of states of M and $f(n)$ be a function defined as

$$f(n) = \sqrt{\frac{n \log n}{T(n)}} \tag{2.1}$$

for all $n \geq 2$. Note that f is strictly increasing by definition, that is, $T(n)$ grows slower than $n \log n$ during the computation. Choose the smallest number $c \in \mathbb{N}$ such that for every $n \geq 2$,

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

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

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

Consider all critical-boundaries of x_0 along path s_0 that have crossing sequences of lengths less than $\log n_0 / f(n_0)$. Let h be the number of all such critical-boundaries. Since the total computation steps in s_0 is equal to the sum of the lengths of any crossing sequences of every inter-cell boundary, we have

$$T(n_0) > c + (n_0 - h)\frac{\log n_0}{f(n_0)}.$$

The inequality comes from the assumption that the length of γ_0 is longer than c. Thus we have

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

which is equal to the number of all crossing sequences of lengths smaller than $\log n_0 / f(n_0)$. Hence there exist at least four distinct critical-boundaries b_1, b_2, b_3, b_4 that have an identical crossing sequence in path s_0. Thus at least two of them, say b_1 and b_2, are on the same side of b_0. Now we remove 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 computation path of M on input x_0' and still has a crossing sequence whose length is greater than c. This contradicts the minimality of x_0. This completes the proof. □

Hennie [33] proved that any deterministic computation with crossing sequences of constant length recognizes only regular languages. We generalize his result to the nondeterministic case. Our proof relies on Lemma 1.2.7.

Lemma 2.2.8 ([51]) *Let L be a language over alphabet Σ and M be a 1NTM that recognizes L. Assume that there exists a constant $c \in \mathbb{N}$ such that for every $x \in L$, the length of any crossing sequence at each inter-cell boundary in each accepting computation path of M on input x is at most c. Then L is regular.*

Proof. Let S be the set of sequences of states of M whose lengths are at most c. For each $x \in \Sigma^*$ and each $v \in S$, we say that x *supports* v if there exists $z \in \Sigma^*$ such that $xz \in L$, and v is the crossing sequence at the inter-cell boundary between x and z on some accepting computation path of M on input xz. For each $x \in \Sigma^*$, let

$$\mathrm{Sup}(x) = \{v \in S \mid x \text{ supports } v\}. \tag{2.2}$$

Note that

$$\forall x, y, z \in \Sigma^* \, [\mathrm{Sup}(x) = \mathrm{Sup}(y) \Longrightarrow \mathrm{Sup}(xz) = \mathrm{Sup}(yz)]. \tag{2.3}$$

and

$$\forall x, y \in \Sigma^* \, [x \in L \wedge \mathrm{Sup}(x) = \mathrm{Sup}(y) \Longrightarrow y \in L]. \tag{2.4}$$

We now define the equivalence relation E on Σ^* as follows:

$$\forall x, y \, [xEy \iff \mathrm{Sup}(x) = \mathrm{Sup}(y)].$$

Since $\mathrm{Sup}(x)$ is a subset of the finite set S for every $x \in \Sigma^*$, the first condition in Lemma 1.2.7 holds. Furthermore ,

$$\begin{aligned}
xEy &\iff Sup(x) = Sup(y) \\
&\Longrightarrow \forall z \in \Sigma^* \, [Sup(xz) = Sup(yz)] \qquad \text{(by Equation 2.3)} \\
&\Longrightarrow xz \in L \iff yz \in L \\
&\qquad \text{(by Equation 2.4 and the definition of support)} \\
&\iff xR_L y.
\end{aligned}$$

Therefore, the second condition in Lemma 1.2.7 holds. Hence L is regular. □

Theorem 2.2.9 ([51]) *Let $T(n) = o(n \log n)$. Then*

$$\mathbf{REG} = \text{1-}\mathbf{NTime}(T(n)) = \text{co-1-}\mathbf{NTime}(T(n)).$$

Proof. $\mathbf{REG} \subseteq \text{1-}\mathbf{NTime}(T(n))$ is obvious since $\text{1-}\mathbf{DTime}(T(n)) \subseteq \text{1-}\mathbf{NTime}(T(n))$. It follows from Lemmas 2.2.7 and 2.2.8 that $\text{1-}\mathbf{NTime}(T(n)) \subseteq \mathbf{REG}$. Thus we have $\mathbf{REG} = \text{1-}\mathbf{NTime}(T(n))$. The second equality follows from the fact that **REG** is closed under complementation (see Lemma 1.2.6). Therefore, $\mathbf{REG} = \text{co-}\mathbf{REG} = \text{co-1-}\mathbf{NTime}(T(n))$. □

This implies that nondeterminism/co-nondeterminism does not gain any advantage over deterministic computations under the one-tape $o(n \log n)$-time restriction. The following corollary is an immediate consequence of Theorem 2.2.9.

Corollary 2.2.10 ([51]) *Let L be a non-regular language and M be a $T(n)$-1NTM that recognizes L. Then there exists a constant $c > 0$ such that for infinitely many n, $T(n) \geq cn \log n$.*

In other words, the running time for any 1NTM recognizing a non-regular language is at least $O(n \log n)$.

2.2.4 Communication Complexity

The concept of communication complexity was first introduced by Yao [76] in 1979. Yao's model can be generalized as the following scenario. Assume that there are two parties, say Alice and Bob, who on given some fixed input information, try to perform some task (that is, to calculate some function) by communicating to each other. Communication complexity is a measure of the number of communications needed to achieve the goal such that the computational power for each party is not considered.

Definition 2.2.11 ([16]) For a given language L over alphabet Σ, $M_L^{(n)}$ denotes the $r_n \times r_n$ 0-1 matrix whose rows and columns are indexed by strings in $\Sigma^{\leq n}$ such that every (x, y)-entry $M_L^{(n)}[x, y]$ satisfies:

$$M_L^{(n)}[x, y] = 1 \iff xy \in L$$

where $r_n = |\Sigma^{\leq n}|$. Let $b \in \{0, 1\}$. A *b-tiling* is a sub-matrix of $M_L^{(n)}$ specified by a pair (R, C) where $R, C \subseteq \Sigma^{\leq n}$, and all entries have value b. The *size* of a b-tiling is the number of its entries; that is, $|R| \cdot |C|$. The *nondeterministic communication complexity* of L at n, denoted $N_L^1(n)$, is the minimal size of a 1-tiling of $M_L^{(n)}$. The sum $N_L^0(n) + N_L^1(n)$ of L at n is simply denoted $N_L(n)$.

Figure 2.7 demonstrates a 1-tiling such that the middle 1 can be included in both squares.

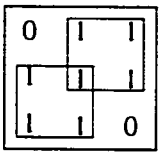

Figure 2.7: A 1-tiling

Definition 2.2.12 Let L be any language and M be a 1DTM. We say that M recognizes L if for each input x, there exists a *certificate* y such that

$$x \in L \implies M(\langle x, y \rangle) = 1, \text{ and}$$
$$x \notin L \implies M(\langle x, y \rangle) \neq 1$$

where a certificate is a binary string of length $\log(T(n))$. More precisely, it is a series of nondeterministic choices of a "correct" path. If $x \in L$, the certificate y leads to an accepting state; otherwise, it leads to a non-accepting state.

Lemma 2.2.13 *Definition 2.2.3 ≡ Definition 2.2.12.*

Proof. By definitions. Definition 2.2.3 includes Definition 2.2.12 since the certificate is only one of paths in a nondeterministic computation.

Let L be any language and N be a $T(n)$-1NTM that recognizes L. Let M be a 1DTM. Then for any string x and some constant c, let y be a certificate of length $\log(T(n)) = \log(cn)$ such that M take $\langle x, y \rangle$ as input and performs the computation according to y. Note that y is the certificate from the computation of N that always leads to the correct answer.

Claim 2 *M simulates N according to y in $T(n)$ steps.*

Proof of Claim 2. Since y is a correct computation path of N, M accepts whenever there is an accepting path in N's computation and rejects if all N's computation paths reject. Note that the encoding $\langle x, y \rangle$ has length of $c' \cdot |xy| = c' \cdot (n + \log n)$ for some constant c' and the expected number of "visits" to each inter-cell boundary is some constant K; therefore, the total running time is $K \cdot c \cdot (n + \log n) = K'n$ for some constant K'. □

By Claim 2, Definition 2.2.12 includes Definition 2.2.3, and this completes the proof. □

The following lemma was stated in [47]. We modify the proof for probabilistic communication complexity and prove the following lemma.

Lemma 2.2.14 *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be a function M be a 1NTM that runs in time $T(n)$ on inputs of the form $x0^n y$ of size $m = 3n$ where $|x| = |y|$. Assume that L is the set of such strings. If M accepts all inputs such that $f(x, y) = 1$, and rejects all inputs such*

that $f(x, y) = 0$. *then the nondeterministic communication complexity to compute* f. $N^1_L(f)$. *is* $O((T(n))/n + \log n)$.

Proof. Let M be a 1NTM and z be a certificate defined by Definition 2.2.12. Alice and Bob. on input (x, y). simulate M on input $x0^n y$ using some nondeterministic protocol. Assume that both Alice and Bob reads the value of z. which represents the location of 0-region or the channel. At each round. Alice and Bob communicate through the location specified by the certificate. Alice simulates M whenever the head is on the left of this location. and Bob simulates M whenever the head is on the right of this location. Each time the head crosses this location. only the state of the finite control needs to be sent. Since $T(n)$ be the total running time. the expected number of crossings on each location is $T(n)/n$. At each location. $N^1(n) - \log n$ messages are sent where $\log n$ is the length of z. Thus

$$N^1(n) = \frac{T(n)}{n} + \log n.$$

Therefore. the expected number of communications is $O(T(n)/n + \log n)$. □

2.2.5 co-1-NTime($O(n \log n)$)

In this section. we consider the nondeterministic computation and the co-nondeterministic computation with broader time bound. All the results in this section are shown by the author.

Lemma 2.2.15 $L_{pal} \in$ **1-NTime**($\Theta(n^2)$).

Proof. To obtain the lower bound we use Lemma 2.2.14. Let $u = x0^n y$ be an input of size $m = 3n$ where $|x| = |y| = n$. Then we can formulate a $2^n \times 2^n$ matrix shown as Fig 2.8. Since $N^1(f) = \log 2^n = n$. by Lemma 2.2.14. we have $N^1(f) = O((T(n))/n + \log n) = n$. Hence the running time $T(n) = (n - \log n) \cdot n = c \cdot n^2$. Therefore. the lower bound for the running time is $\Omega(n^2)$.

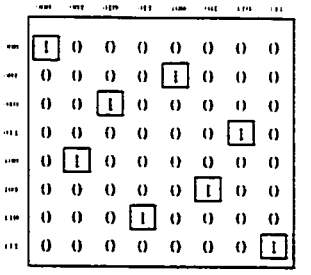

Figure 2.8: The 1-tiling for palindrome where $|x| = |y| = 3$

The upper bound can be obtained by using the same argument as in Lemma 2.1.16. Therefore. $T(n) = \Omega(n^2) = O(n^2) = \Theta(n^2)$. □

The following lemma is the main result in this section.

Lemma 2.2.16 ([51]) $L_{pal} \in$ co-**1-NTime**$(\Theta(n \log n))$.

Proof Sketch. The proof consists of two essential ideas:

1. If we observe the nondeterministic and co-nondeterministic computations, notice that the accepting criteria for a nondeterministic computation holds when at least one of the computation paths returns 1 where on the other hand, a co-nondeterministic computation requires all the computation paths to return 1 (Figure 2.9). Thus if one of the paths returns 0 for a co-nondeterministic computation tree, the input will be rejected.

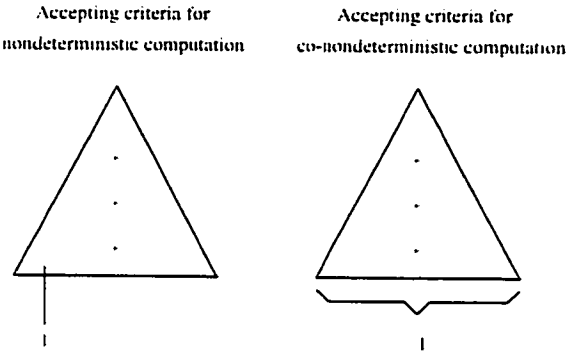

Figure 2.9: The accepting criteria for nondeterministic and co-nondeterministic computations

   Our algorithm takes the fact that if a string is not a palindrome, there exists an index i such that $x_i \neq y_{n-i}$ where x_i and y_{n-i} are the ith symbol of string x and $(n-i)$th symbol of y.

2. Each path consists of a series of nondeterministic choices; each of which has the size of $\log n$ where n is the length of input. Imagine each character in a given string as a representation of even or odd. Then we can find the position indicated from the given series. Figure 2.10 demonstrates the essential idea of the algorithm. Suppose that a machine is given a binary string 0101; that is, position indexed 5. Whenever it reads a 1 (0, respectively), it crosses out all the odd-th (even, respectively) cells. At the step 3, the machine find the desired position.

■

Proof. Since $L_{pal} \notin$ **REG**, $T(n) = \Omega(n \log n)$. To show the upper bound, we construct a 1NTM M as follows:

   write the input on the tape and place the head at the start cell

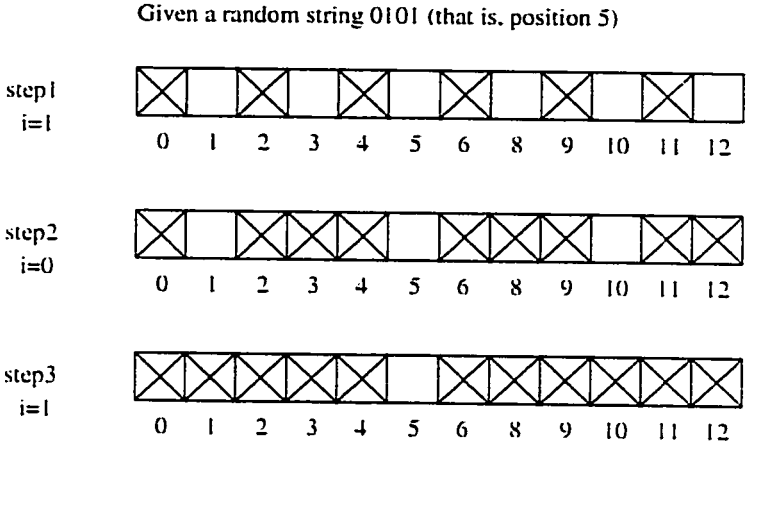

Figure 2.10: An example on each deterministic path of a co-nondeterministic computation

by scanning once from left to right, check if w is of the form $x\#y$

if w is not of such a form, reject

else

loop

from left to right, check the current lengths of x and y

if $|x| = 1$ or $|y| = 1$, exit the loop

else

move the head back to the start cell

choose in a co-nondeterministic manner: $i = 1$ or $i = 0$

if $i = 1$

on x-region, from left to right, overwrite all the odd-th symbols by $\diamond$ and stop at $\#$

place the head on the rightmost cell of y

on y-region, from right to left, overwrite all the odd-th symbols by $\diamond$ and stop at $\#$

place the head back to the start cell

else (if $i = 0$)

on x-region, from left to right, overwrite all the even-th symbols by $\diamond$ and stop at $\#$

place the head on the rightmost cell of y

on y-region, from right to left, overwrite all the even-th symbols by $\flat$ and stop at #

place the head back to the start cell

end loop

check the lengths of x and y

if $|x| > 1$ or $|y| > 1$, reject

check if the symbol left in x-region is the same as the symbol left in the y-region

if they are the identical, accept

else reject

According to our algorithm, it is easy to see that if the lengths of x and y are different, M rejects. Now we consider the following two cases where $|x| = |y|$.

Case1: If $x \in L_{pal}$, all computation paths of M will accept. This is true since, on each computation path, each remaining pair of x and y must be the same.

Case2: If $x \notin L_{pal}$, at least one of computation paths of M will reject. This follows the fact that one of the remaining pair of x and y must be different.

For the running time analysis, we verify the following steps. Checking the format of the input string takes linear time. It also takes linear time to check if $|x| = 1$ or $|y| = 1$. At each iteration of the loops, M overwrites half of the symbols on each of x-region and y-region. This, in total, takes $O(n \log n)$ time since $|w| = 2n + 1$. Finally, checking the length of x and y takes linear time, and checking if the remaining bit is identical takes at most linear time. Thus the total running is $O(n \log n)$.

In conclusion, $L_{pal} \in$ co-**1-NTime**$(O(n \log n))$: therefore, $T(n) = \Omega(n \log n) = O(n \log n) = \Theta(n \log n)$. □

Similarly, we show the following proposition.

Proposition 2.2.17 $\overline{L_{pal}} \in$ **1-NTime**$(O(n \log n))$, *and*

$\overline{L_{pal}} \notin$ co-**1-NTime**$(O(n \log n))$

Since **1-NTime**$(O(n \log n)) \neq$ co-**1-NTime**$(O(n \log n))$, the following theorem applies.

Theorem 2.2.18 **1-NTime**$(O(n \log n))$ *is not closed under complementation.*

By combining the results shown previously, we conclude the following theorem.

Theorem 2.2.19 *Let $T(n) = O(n \log n)$. Then*

1. **1-DTime**$(T(n)) \subsetneq$ **1-NTime**$(T(n))$
2. **1-DTime**$(T(n)) \subsetneq$ co-**1-NTime**$(T(n))$

Proof.

1. By Proposition 2.1.15 and Proposition 2.2.17.
2. By Lemma 2.1.14 and Lemma 2.2.16.

□

2.3 Set-theoretical Difference of Two Languages

The class $\mathbf{D^P}$ was first introduced by Papadimitriou and Yannakakis. The word "D" stands for "difference", which is the set-theoretical difference between languages in **NP**. A common misconception is that one might think $\mathbf{D^P}$ is equal to $\mathbf{NP} \cap$ co-**NP**. In fact, it is believed that $\mathbf{D^P}$ might not be contained even in $\mathbf{NP} \cup$ co-**NP**. We give a one-tape linear-time definition as follows.

Definition 2.3.1 Let L be a language. L is in **1-D$^\mathbf{P}$Time**$(o(n \log n))$ if and only if there exists a language $L_1 \in$ **1-NTime**$(o(n \log n))$ and a language $L_2 \in$ co-**1-NTime**$(o(n \log n))$ such that $L = L_1 \cap L_2$.

Lemma 2.3.2 **REG = 1-D$^\mathbf{P}$Time**$(o(n \log n))$.

Proof. Let L be any language in **1-D$^\mathbf{P}$Time**$(o(n \log n))$. By definition, there must exist languages $L_1 \in$ **1-NTime**$(o(n \log n))$ and $L_2 \in$ co-**1-NTime**$(o(n \log n))$ such that $L = L_1 \cap L_2$.

We prove the following lemma.

Lemma 2.3.3 **REG** *is closed under intersection.*

Proof of Lemma 2.3.3. Let L_1 and L_2 be languages recognized by some DFA M_1 and M_2, respectively. We construct a $T(n)$-1DTM M as follows:

input $w = \langle x, \langle M_1 \rangle, \langle M_2 \rangle \rangle$

simulate M_1 on x

if M_1 rejects, reject and halt

else M simulate M_2 on x

if M_2 accepts, accept and halt

else reject and halt

By the algorithm, it is clear that M accepts only when both M_1 and M_2 accept. Thus M recognizes $L_1 \cap L_2$. Note that $|\langle M_1 \rangle|$ and $|\langle M_1 \rangle|$ are constants; thus the running time $T(n) = cn$ where c is some constant. Since **1-DTime**$(O(n))$ = **REG**. $L_1 \cap L_2 \in$ **REG**. □

Since all languages in **1-NTime**$(o(n \log n))$ and co-**1-NTime**$(o(n \log n))$ are regular, by Lemma 2.3.3. L is regular. Hence **1-D$^{\mathbf{P}}$Time**$(o(n \log n))$ = **REG**. □

Chapter 3

Probabilistic and Counting Classes

3.1 Probabilistic Computations

3.1.1 Background

A variant of Turing machines, probabilistic Turing machine, was introduced by DeLeeuw, Moore, Shannon, and Shapiro in 1956. It is a Turing machine with the ability to "coin-flip" before each computation step. Surprisingly, this model gives solutions to some problems that are believed to be unsolvable using deterministic Turing machines. In 1977 Gill introduced complexity class **PP** (probabilistic polynomial) and showed that **PP** is closed under complementation. Ever since the power of probabilistic computation were discovered, researchers have tried to discover more of its properties. In 1995 Beigel, Reingold, and Spielman showed the closure properties of **PP** under union. By DeMorgan's Law, since **PP** is closed under union and complementation, it is also closed under intersection.

3.1.2 Probabilistic Turing Machines

The formal definition of a one-tape probabilistic Turing machine is given as follows.

Definition 3.1.1 (one-tape probabilistic Turing machines) A one-tape probabilistic Turing machine (or 1PTM for short) M is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F_{acc}, F_{rej})$ where

Q is a finite set of internal states,

Σ is a finite alphabet with a distinguished blank symbol $\{\sqcup\} \notin \Sigma$,

Γ is a tape alphabet with $\Sigma \cup \{\sqcup\} \subseteq \Gamma$,

δ is a transition function: $(Q - F_{acc} - F_{rej}) \times \Sigma \longrightarrow \tilde{\mathbb{R}}^{\Sigma \times Q \times \{L,N,R\}}$,

q_0 is the initial state,

$F_{acc} \subset Q$ is a set of accepting states, and

$F_{rej} \subset Q$ is a set of rejecting states where $F_{rej} \neq F_{qcc}$.

We define one-tape $T(n)$-time probabilistic Turing machines as follows.

Definition 3.1.2 ($T(n)$-time 1PTM) Let $T:\mathbb{N} \to \mathbb{N}$. A $T(n)$-time 1PTM (abbreviated $T(n)$-1PTM) is a one-tape probabilistic Turing machine such that, for each input x, there are exactly two probabilistic choices at each step in a non-final configuration, and the length of every computation path of M on input x is at most $T(|x|)$.

In Definition 3.1.2, we do not require that for every x, all of computations of a $T(n)$-1PTM on input x halt after the same number of steps. In Figure 3.1, the probability on each node is the product of probabilities on proceeding edges where each edge represents a probabilistic choice with probability a half.

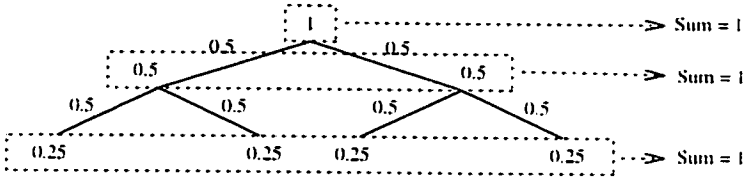

Figure 3.1: A probabilistic computation tree.

Definition 3.1.3 (accepting probability) Let $T:\mathbb{N} \to \mathbb{N}$ and M be a $T(n)$-1PTM. We write $M(x)$ for the set of all accepting computation paths of M on input x. Here the length of a computation path is the number of applications to the transition function along the path. The accepting probability of M on input x is denoted by $P_a^M(x)$ and is defined as

$$P_a^M(x) \equiv \sum_{s \in M(x)} \left(\frac{1}{2}\right)^{|s|}. \tag{3.1}$$

3.1.3 Bounded-Error Probabilistic Computations

The class **BPP** was first introduced by Gill and has been attracting much attention. If we consider making a probabilistic computation as flipping a fair coin, bounded-error probabilistic computation is as flipping a biased coin (Figure 3.2). A rather interesting result, $\mathbf{BPP} \subseteq \mathbf{PP}$, implies that a biased-coin flip does not give a machine extra computation power over a fair-coin flip. Interestingly, the equivalence of **BPP** and **NP** remains open.

Definition 3.1.4 (bounded-error probability) Let L be a language. We say that M recognizes L with bounded error probability if there exists an $\varepsilon > 0$ such that

$$\begin{aligned} x \in L &\implies P_a^M(x) \geq 1/2 + \varepsilon, \text{ and} \\ x \notin L &\implies P_a^M(x) \leq 1/2 - \varepsilon. \end{aligned}$$

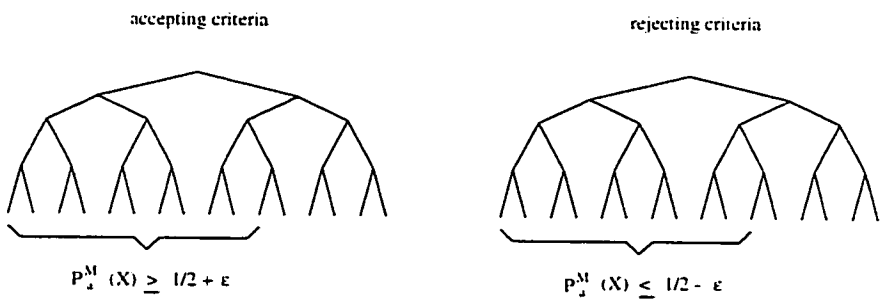

Figure 3.2: A schematic for bounded-error probabilistic computation tree.

Definition 3.1.5 Let $T:\mathbb{N} \to \mathbb{N}$. The class **1-BPTime**$(T(n))$ is defined as the set of languages L such that there exists a $T(n)$-1PTM that recognizes L with bounded-error probability.

In this section, we prove **1-BPTime**$(o(n \log n)) =$ **REG** using a crossing sequence argument. This further implies the equivalence of **1-NTime**$(o(n \log n))$, co-**1-NTime**$(o(n \log n))$, and **1-BPTime**$(o(n \log n))$.

Recall the proof for Lemma 2.1.6, we can separate a computation at the given boundary. We formally define the left-end and right-end configurations as follows.

Definition 3.1.6 Let $T:\mathbb{N} \to \mathbb{N}$ and $M = (Q, \Sigma, \Gamma, \delta, q_0, F_{acc}, F_{rej})$ be a $T(n)$-1PTM. A right-end configuration of M is a 3-tuple: the contents of the segment on the tape that has a right-end and infinite cells to the left, the location of the tape head and the state $q \in Q$ of the finite control of M. Similarly, a left-end configuration of M is a 3-tuple: the contents of the segment on the tape that has a left-end and infinite cells to the right, the location of the tape head and the state $q \in Q$ of the finite control of M. Let $x = x_L x_R \in \Sigma^*$ be an input and $v = v_1 v_2 \ldots v_m \in Q^*$ be a crossing sequence where $v_i \in Q$. We say that a finite sequence of right-end configurations $C_0, C_1, \ldots, C_n$ of M is consistent with v from the left on x (on segment t_L), if the following conditions hold:

1. Initially, M starts with configuration C_0 where the tape content is x_L (the initial content of t_L) ending at the rightmost cell and is filled with ▷ elsewhere, the tape head is in the leftmost cell of t_L, and the current state is q_0.

2. There exists an increasing function $f:\{1, 2, \ldots, \lceil m/2 \rceil\} \to \{0, 1, \ldots, n\}$ that satisfies the following conditions:

   (i) For each k, the tape content is $C_{f(k)}$, the tape head is in the rightmost cell of the tape, and the state is v_k. For each k, there exists a probabilistic choice from $C_{f(k)}$ by which the tape head leaves the right-end of t_L to the right, the tape content changes from $C_{f(k)}$ to $C_{f(k)+1}$ (in the case where $f(k)+1 \leq n$), and the state of M changes into v_{2k-1}. Similar to the proof of Lemma 2.1.6, the head is pushed back. For each k, if $f(k)+1 \leq n$, then $2k$ is less or equal to m, the tape head is in the rightmost cell of t_L, the tape content stays $C_{f(k)+1}$, and the state of $C_{f(k)+1}$ is v_{2k}.

(ii) For each i, if $i \neq f(k)$ for every k and $i < n$, then C_{i+1} is obtained by applying the transition function δ to C_i in an obvious way.

3. If m is even, the state of C_n is in F_{acc}.

This definition of "consistency from the left" works in the case where $x = \epsilon$. Recall the proof of Lemma2.1.6, we define our machine to enter the accepting state if and only if the head stops at the right of any given inter-cell boundary. In that case, we create one extra crossing for each inter-cell boundary when an empty string ϵ is in the set; otherwise, the head stays on the left of some inter-cell boundary. Since 0 is considered to be even and 1 to be odd, the definition given above holds.

We say a finite sequence of left-end configurations $C_0, C_1, \ldots, C_n$ of M is consistent with v from the right on x (on segment t_R) if the following conditions hold:

1. Initially, M starts with configuration C_0 where the tape content is x_R (the initial content of t_R) beginning from the leftmost cell and is filled with $\flat$ elsewhere, the tape head is in the leftmost cell of segment t_R, and the current state is v_1.

2. There exists an increasing function $f: \{1, 2, \ldots, \lceil (m-1)/2 \rceil\} \to \{0, 1, \ldots, n\}$ that satisfies the following conditions:

   (i) For each k, the tape content is $C_{f(k)}$, the tape head is in the leftmost cell of the tape, and the state is v_k. For each k, there exists a probabilistic choice from $C_{f(k)}$ by which the tape head leaves the left-end of t_R to the left, the tape content changes from $C_{f(k)}$ to $C_{f(k)+1}$ (in the case where $f(k)+1 \leq n$), and the state of M changes into v_{2k}. Similar to the proof of Lemma 2.1.6, the head is pushed back. For each k, if $f(k)+1 \leq n$, then $2k+1$ is less or equal to m, the tape head is in the rightmost cell of t_R, the tape content stays $C_{f(k)+1}$, and the state of $C_{f(k)+1}$ is v_{2k+1}.

   (ii) For each i, if $i \neq f(k)$ for every k and $i < n$, then C_{i+1} is obtained by applying the transition function δ to C_i in an obvious way.

3. If m is odd, the state of C_n is in F_{acc}.

The definition of "consistency from the right" also holds for the case where $x = \epsilon$. The whole schematic for the right-end and left-end configuration is shown in Figure 3.3.

Next, we write $\mathcal{CONS}_L^M(x; v)$ for the set of finite sequences of right-end configurations of M that are consistent with v from the left on x and $\mathcal{CONS}_R^M(v; x)$ for the set of finite sequences of left-end configurations of M that are consistent with v from the right on x. Finally, we define accepting probability of left-end configuration $\mathcal{W}_{\mathcal{L}}{}^M(x; v)$ and the accepting probability of right-end configuration $\mathcal{W}_{\mathcal{R}}{}^M(v; x)$ by

$$\mathcal{W}_{\mathcal{L}}{}^M(x; v) \equiv \sum_{s \in \mathcal{CONS}_L^M(x; v)} \left(\frac{1}{2}\right)^{|s|}, \text{ and}$$

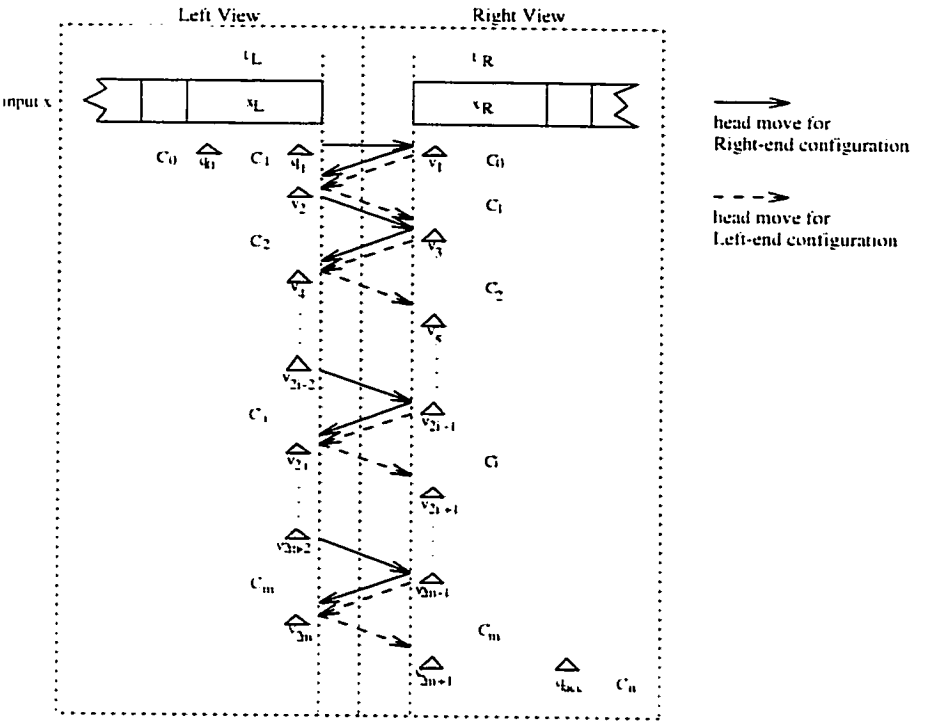

Figure 3.3: A schematic for right-end/left-end configurations

$$\mathcal{W}_{\mathcal{R}}{}^{M}(v{:}x) \equiv \sum_{s \in C.VS_{R}^{M}(v,x)} \left(\frac{1}{2}\right)^{|s|}$$

where $|s|$ is the number of applications of the transition function δ along s. In other words, $|s|+1$ is the number of right-end (left-end, respectively) configurations in s.

Remark 3.1.7 Let M be a $T(n)$-1PTM. Then for each $x \in \Sigma^*$ and each $v \in Q^*$,

$$0 \leq \mathcal{W}_{\mathcal{L}}{}^{M}(x{:}v).\mathcal{W}_{\mathcal{R}}{}^{M}(v{:}x) \leq 1. \tag{3.2}$$

Theorem 3.1.8 ([51]) *Let L be a language over Σ and M be a $T(n)$-1PTM. Suppose that M recognizes L with bounded-error probability. If there exists $c \in \mathbb{N}$ such that for every $x \in L$, the length of crossing sequence of every inter-cell boundary in every computation path of M on input x is at most c, L is regular.*

Proof. Let S be the set of sequences of states of M whose lengths are at most c. Since M recognizes L with bounded-error probability, there exists an $\varepsilon > 0$ such that

$$\begin{aligned} x \in L &\implies P_a^M(x) \geq 1/2 + \varepsilon, \text{ and} \\ x \notin L &\implies P_a^M(x) \leq 1/2 - \varepsilon. \end{aligned}$$

There are several ways to bound $\mathcal{W}_{\mathcal{L}}{}^{M}(x;v)$. The author takes the first squadron of the unit circle and divides it into n segments to obtain a new bound that is different from [51].

Remark 3.1.9 For any $\alpha \in [0.1/2]$, $0 \leq \sin(\alpha\pi) \leq 1$, and the interval between is increasing (see Figure 3.4).

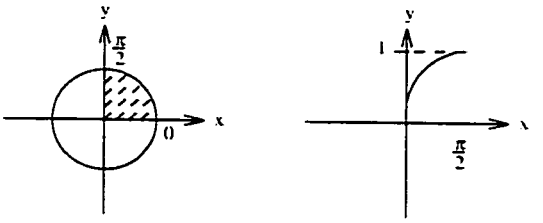

Figure 3.4: Unit circle.

We choose $n \in \mathbb{N}^{+}$ such that $|S| \cdot \sin(\frac{1}{2n}\pi) < 2\varepsilon$ and bound $\mathcal{W}_{\mathcal{L}}{}^{M}(x;v)$ as follows. For each $x \in \Sigma^{*}$, each i with $1 \leq i \leq n$, and each $v \in S$, we say that x supports (i,v) if

$$\sin(\frac{i-1}{2n}\pi) \leq \mathcal{W}_{\mathcal{L}}{}^{M}(x;v) \leq \sin(\frac{i}{2n}\pi). \tag{3.3}$$

For each $x \in \Sigma^{*}$, let

$$\mathrm{Sup}(x) = \{(i,v) \mid 1 \leq i \leq n \ \wedge \ v \in S \ \wedge \ x \text{ supports } (i,v)\}. \tag{3.4}$$

We define an equivalence relation E on Σ^{*} by

$$xEy \overset{def}{\Longleftrightarrow} \mathrm{Sup}(x) = \mathrm{Sup}(y).$$

Since $\mathrm{Sup}(x)$ is a subset of the finite set $\{1,\ldots,n\} \times S$ for every $x \in \Sigma^{*}$, the first condition in Lemma 1.2.7 holds for E.

Note that for every $x, z \in \Sigma^{*}$,

$$P_{a}^{M}(xz) = \sum_{v \in S} \mathcal{W}_{\mathcal{L}}{}^{M}(x;v)\mathcal{W}_{\mathcal{R}}{}^{M}(v;z). \tag{3.5}$$

Remark 3.1.10 Since we divide $[0.1/2]$ into n segments, every segment will be equal in size, and the last segment will be equal to or less than the average size of segment.

Suppose that xEy. By Remark 3.1.10, if $i = 1$, $0 \leq \mathcal{W}_{\mathcal{L}}{}^{M}(x;v) \leq \sin(\frac{1}{2n}\pi)$. Thus, for every $v \in S$,

$$|\mathcal{W}_{\mathcal{L}}{}^{M}(x;v) - \mathcal{W}_{\mathcal{L}}{}^{M}(y;v)| \leq \sin(\frac{1}{2n}\pi).$$

Hence, for each $z \in \Sigma^{*}$,

$$\begin{aligned} |P_n^M(xz) - P_n^M(yz)| &\leq \sum_{v \in S} |\mathcal{W}_{\mathcal{L}}{}^M(x; v) - \mathcal{W}_{\mathcal{L}}{}^M(y; v)| \mathcal{W}_{\mathcal{R}}{}^M(v; z) \\ &\leq \sum_{v \in S} |\mathcal{W}_{\mathcal{L}}{}^M(x; v) - \mathcal{W}_{\mathcal{L}}{}^M(y; v)| \\ &\leq |S| \cdot \sin(\frac{1}{2n}\pi) \\ &< 2\varepsilon. \end{aligned}$$

It follows that $xz \in L$ if and only if $yz \in L$; therefore, xR_Ly is properly defined. Thus the second condition in Lemma 1.2.7 holds for E. Hence L is regular. □

Using Theorem 3.1.8. we prove the following theorem.

Theorem 3.1.11 ([51]) *Let $T: \mathbb{N} \to \mathbb{N}$. Suppose that $n + 1 \leq T(n)$ and $T(n) = o(n \log n)$. Then*

$$\textbf{REG} = \textbf{1-BPTime}(T(n)).$$

Proof. $\textbf{REG} = \textbf{1-DLIN} \subseteq \textbf{1-BPTime}(T(n))$ is trivial. It follows from Lemma 2.2.7 and Theorem 3.1.8 that $\textbf{1-BPTime}(T(n)) \subseteq \textbf{REG}$. Thus the result is obtained. □

We have the following from Theorem 3.1.11.

Corollary 3.1.12 ([51]) *Let $T: \mathbb{N} \to \mathbb{N}$ and M be a $T(n)$-1PTM. Suppose that L is a non-regular language and M recognizes L with bounded-error probability. Then there exists $c > 0$ such that, for infinitely many n, $T(n) \geq cn \log n$.*

The following Theorem follows immediately from Theorem 3.1.11.

Theorem 3.1.13 ([51]) $\textbf{REG} = \textbf{1-BPLIN}$.

3.2 Counting

3.2.1 Background

The counting class $\mathbf{C_=P}$ was first introduced by Wagner [72]. This class can be defined using counting machines or gap functions. The main characteristic is that, instead of checking the existence of an accepting path among a nondeterministic computation, a machine or a function checks if the number of accepting paths is equal to the number of rejecting paths. Similar to **NP**, the complementation property for this class remains open.

3.2.2 Models and Definitions

Generalized 1-C$_=$LIN

First we define the class **1-C$_=$LIN** using counting machines. Intuitively, a counting machine is a nondeterministic Turing machine that does a extra step at the end of computation: that is, it returns the difference between the accepting paths and the rejecting paths. We use it to define the generalized **1-C$_=$LIN**. Figure 3.5 shows the computation tree of **1-C$_=$LIN**.

Definition 3.2.1 (one-tape $T(n)$-time CM) Let $T:\mathbb{N} \to \mathbb{N}$. A one-tape $T(n)$-time counting machine (abbreviated $T(n)$-1CM) is a 1NTM M such that for each input x, M counts and returns $|M(x)| - |\overline{M}(x)|$ at the end of computation, and every computation path of M on input x is at most $T(|x|)$.

Definition 3.2.2 Let L be a language and let $T:\mathbb{N} \to \mathbb{N}$. We say that a $T(n)$-1CM M recognizes L if for every x,

$$\begin{aligned} |N(x)| &= |\overline{N}(x)| \Longrightarrow x \in L, \text{ and} \\ |N(x)| &\neq |\overline{N}(x)| \Longrightarrow x \notin L. \end{aligned}$$

The class **1-C$_=$Time**($T(n)$) is defined as the set of languages L such that there exists a $T(n)$-1CM that recognizes L. Then the class **1-C$_=$LIN** is defined by

$$\textbf{1-C}_=\textbf{LIN} \equiv \textbf{1-C}_=\textbf{Time}(O(n)). \tag{3.6}$$

Therefore, **1-C$_=$LIN** is the set of languages recognized by one-tape linear-time counting machines.

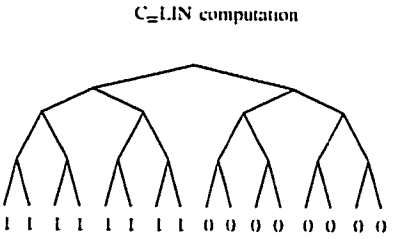

Figure 3.5: Computation tree for **1-C$_=$LIN**

Synchronized 1-C$_=$LIN

Similar to counting machines, a gap function also counts the difference between the the number of accepting paths and the number of rejecting paths of a nondeterministic computation. However, the definition of **1-C$_=$LIN** given using such a function is *synchronized*, which means

that a computation tree forms a complete binary tree. It is still an open question whether the synchronized **1-C$_=$LIN** is equal to the generalized **1-C$_=$LIN**.

Definition 3.2.3 (1-GapLIN) Given any $L \in \Sigma^*$ over alphabet Σ. Let $L_x = \{y \in \Sigma^* \mid \langle x, y \rangle \in L\}$. A function $f : \{0,1\}^* \to \mathbb{Z}$ is in **1-GapLIN** if there is a language L in **1-DLIN** and a constant c such that

$$f(x) = \frac{|\Sigma^{c \cdot n} \cap L_x| - |\Sigma^{c \cdot n} - L_x|}{2}$$

where n is the length of x. Moreover, since we previously prove that **1-DLIN = REG**, the language L is regular.

Definition 3.2.4 A language L is said to be in the class **1-C$_=$LIN** if there is a **1-GapLIN** function f such that for any x, $x \in L$ if and only if $f(x) = 0$. The class co-**1-C$_=$LIN** is the set of all languages not in **1-C$_=$LIN**

3.2.3 A Non-Regular Language in 1-C$_=$LIN

The following lemma shows that **1-C$_=$LIN** recognizes a non-regular set.

Lemma 3.2.5 *Let $L_{=} = \{a^n b^m \mid n = m\}$. $L_{=} \in$ **1-C$_=$LIN**.*

Proof Sketch. The power of **1-C$_=$LIN** contains both nondeterminism and counting. At first glance, it may not be convincing that this class contains more than regular languages since **1-NLIN = co-1-NLIN = REG**. In retrospect, a counting machine for this class returns the difference between the number of accepting paths and the number of rejecting paths. If we visualize all a's to be accepting paths and all b's to be rejecting paths, the machine is, in fact, counting the difference between a's and b's. If the difference calculated is zero, the number of a's and the number of b's are the same. ∎

Proof. We define a 1CM M as follows. The tape of M is formated into two tracks where the first track is used to store the input string, and the second track is initially empty. M is constructed as follows:

write the input x on the first track and move the head back to the start cell

deterministically check if x is of the form $a^n b^m$ for some $n \geq 0$ and $m \geq 0$

if x is not of the required form, reject

else move the head back to the start cell

by scanning the tape once from left to right, perform the following tasks:

nondeterministically choose 0 or 1 for each a and b

record all the nondeterministic choices under the corresponding symbols read

(all the nondeterministic choices for a's and b's are called a-choices and b-choices, respectively).

a scan is said to be a success for a's if

all a-choices are 1's, and at least one of b-choices is 0

a scan is said to be a success for b's if

all b-choices are 1's, and at least one of a-choices is 0

finally, make an extra nondeterministic choice

accept if and only if

1. a success for a's occurs, or
2. neither a success for a's nor a success for b's occurs, and the last nondeterministic choice is a 1.

Let $\#tot(M,x) = |M(x)| + |\overline{M}(x)|$ be the total number of computation paths of M that reaches the final configuration (that is, accept or reject). Let $\#suc(a)$ be the number of successes for a's and $\#suc(b)$ be the number of successes for b's. Then

$$\begin{aligned} \#suc(a) &= \frac{1}{2^n} \cdot \#tot(M,x) \cdot (1 - \frac{1}{2^m}) \\ &= 2^{-n} \cdot 2^{n+m+1} \cdot 2^{-m} \cdot (2^m - 1) \\ &= 2^{m+1} - 2. \end{aligned}$$

Similarly, we have

$$\begin{aligned} \#suc(b) &= \frac{1}{2^m} \cdot \#tot(M,x) \cdot (1 - \frac{1}{2^n}) \\ &= 2^{-m} \cdot 2^{n+m+1} \cdot 2^{-n} \cdot (2^n - 1) \\ &= 2^{n+1} - 2. \end{aligned}$$

Each success for a's and b's consists of two parts: that is, all their own nondeterministic choices must be 1's, and at least one of their "opponent's" choices must be 0. The total number of paths for each success depend on how likely their enemy makes a mistake (that is, return a 0). Thus the numbers of $\#suc(a)$ and $\#suc(b)$ depend on the numbers of b's and a's, respectively.

Moreover, the first accepting condition also holds for the occurrence of a success for b's. The second condition tells our machine to split equally the remaining (beside the successes for a's and the successes for b's) paths into two portions. The result is demonstrated as Figure 3.6.

Note that $\#suc(a) - \#suc(b) = 2^{m+1} - 2^{n+1}$. We have the total number of accepting paths:

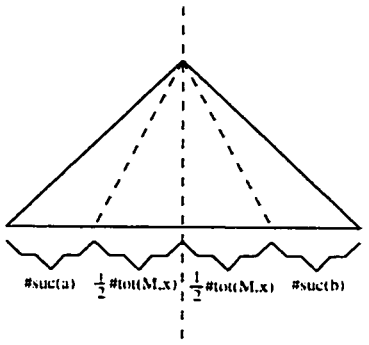

Figure 3.6: Computation tree at the end of the algorithm.

$$\begin{aligned} |M(x)| &= \#suc(a) + (\#tot(M,x) - \#suc(a) - \#suc(b)) \cdot \frac{1}{2} \\ &= \#suc(a) + \frac{1}{2} \cdot \#tot(M,x) - \frac{1}{2} \cdot \#suc(a) - \frac{1}{2} \cdot \#suc(b) \\ &= \frac{1}{2} \cdot \#tot(M,x) + \frac{1}{2} \cdot (\#suc(a) - \#suc(b)). \end{aligned} \tag{3.7}$$

By Equation 3.7, the accepting conditions require M to take $\#suc(a)$ and compare with $\#suc(b)$. The the number of accepting paths is initially equal to the number of rejecting paths. Then the gap varies according to the number of a's and the number of b's.

Case1: If $n = m$.

$$|M(x)| = \frac{1}{2} \cdot \#tot(M,x).$$

Case2: If $n > m$.

$$\begin{aligned} |M(x)| &= \frac{1}{2} \cdot \#tot(M,x) + \frac{1}{2} \cdot (\#suc(a) - \#suc(b)) \\ &= \frac{1}{2} \cdot \#tot(M,x) + \frac{1}{2} \cdot (2^{m+1} - 2^{n+1}) \\ &< \frac{1}{2} \cdot \#tot(M,x). \end{aligned}$$

Case3: If $n < m$.

$$\begin{aligned} |M(x)| &= \frac{1}{2} \cdot \#tot(M,x) + \frac{1}{2} \cdot (\#suc(a) - \#suc(b)) \\ &= \frac{1}{2} \cdot \#tot(M,x) + \frac{1}{2} \cdot (2^{m+1} - 2^{n+1}) \\ &> \frac{1}{2} \cdot \#tot(M,x). \end{aligned}$$

It is clear that checking the correctness of the input format takes linear time as well as making and checking the nondeterministic choices. Therefore, $L_{eq} \in$ **1-C$_=$LIN**.

□

The following proposition is obtained by showing $L_{eq} \notin$ **REG** is in **CFL**.

Proposition 3.2.6 *There exists a CFL in* **1-C$_=$LIN**.

Proof. Lemma A.0.4 shows the non-regularity of L_{eq}. To show that L_{eq} is context-free, we define $G = (V, \Sigma, R, S)$ as follows.

1. $V = \{S\}$.
2. $\Sigma = \{a, b\}$, and
3. R is defined : $S \longrightarrow aSb \mid \epsilon$.

The rest of the proof follows by Lemma 3.2.5. □

By the proof of Lemma 3.2.5, we obtain the following proposition.

Proposition 3.2.7 *Let* $L_{neq} = \{a^n b^m \mid n \neq m \wedge n, m \geq 0\}$. $L_{neq} \in$ co-**1-C$_=$LIN**.

By modifying the language and the proof of Lemma 3.2.5, the following proposition is obtained.

Proposition 3.2.8 *There exists a non-context-free language in* **1-C$_=$LIN**.

Proof. Let $L_{eq3} = \{a^k b^m c^n \mid k = n = m\}$. We want to show that L_{eq3} is not context-free, and $L_{eq3} \in$ **1-C$_=$LIN**.

Lemma A.0.5 shows that L_{eq3} is not context-free. To show that $L_{eq3} \in$ **1-C$_=$LIN**, we simply perform the algorithm in Lemma 3.2.5 twice. □

3.2.4 Closure Property of 1-C$_=$LIN

A probabilistic finite state automaton (or PFA for short) $\mathcal{N}$ is a five tuple $\mathcal{N} = (Q, \Sigma, \pi_0, \{M(x) \mid x \in \Sigma\}, F)$ where

Q is the set of states.

Σ is the set of alphabet.

π_0 is a row vector that represents the initial distribution.

$M(x)$ is a matrix of order $|Q| = k$ for each x, and

$F \subset Q$ is the set of final states.

For each entry $\alpha_{ij}(x)_{i,j\geq 0}$, we assign a probability such that each row sums up to 1 (to form a stochastic matrix) where each α_{ij} represents the probability from state s_i to state s_j.

Let I be the 2×2 identity matrix and $I^{\oplus k}$ be the k dimensional identity matrix. For each input string $u \in \Sigma^*$ where $u = x_0 \cdots x_n$, we define $M(u) = I^{\oplus k}$ if $|u| = 0$; otherwise. $M(u) = M(x_0) \cdots M(x_n)$. Since each $M(x_i)$ is stochastic. $M(u)$ is also stochastic.

The following is a modification of the Definition 3.2.3. Instead of mapping to an integer. the function maps to a real range $[0, 1]$.

Definition 3.2.9 A function $f : \{0,1\}^* \rightarrow [0,1]$ is defined as $f(u) = \pi_0 M(u) \eta^F$ where η^F is a column vector whose i-th component is either 1 if $s_i \in F$; otherwise. it is 0. Let $\mathcal{N}$ be a PFA. The language $L(\mathcal{N}) = \{u \in \Sigma^* \mid f_{\mathcal{N}}(u) = \tau\}$ is in **1-C$_=$LIN** if $\tau = \frac{1}{2}$.

Since $f_{\mathcal{N}}$ calculates an acceptance probability then $f_{\mathcal{N}}$ calculates $\frac{|\Sigma^{=n} \cap L(\mathcal{N})|}{|\Sigma^{=n}|}$ where n is the length of x.

We obtain the next theorem by following the proof of Theorem 1 in [22].

Theorem 3.2.10 *Assume that* $L \in$ **1-C$_=$LIN**. *Then there exists an* $n \in \mathbb{N}$ *such that for any* $x, y, z \in \Sigma^*$.

$$yz, yxz, \cdots, yx^{n-1}z \in L \implies \forall m \geq 0\, [yx^m z \in L].$$

Proof. Let $L \in$ **1-C$_=$LIN**. Then there exists a PFA $\mathcal{N}$ such that $L = \{u \in \Sigma^* \mid f_{\mathcal{N}}(u) = \frac{1}{2}\}$. Let Q be the set of state for $\mathcal{N}$ of cardinality n. Then the linear transformation $M(x)$ is an $n \times n$ stochastic matrix.

The characteristic polynomial can be written as follows:

$$\begin{aligned} 0 &= 1 \cdot \lambda^n - \alpha_{n-1}\lambda^{n-1} - \cdots - \alpha_1 \lambda - \alpha_0. \text{ or} \\ 1 \cdot \lambda^n &= \alpha_{n-1}\lambda^{n-1} + \cdots + \alpha_1 \lambda + \alpha_0. \end{aligned}$$

Nasu and Honda [59] remarked that the characteristic polynomial of a stochastic matrix has the following form

$$\alpha_{n-1} + \cdots + \alpha_1 + \alpha_0 = 1.$$

Recall that

$$M(x) \cdot v = \lambda \cdot v \text{ and } M(x^n) = \overbrace{M(x) \cdots M(x)}^{n}.$$

We replace each eigenvalue λ^l by $M(x^l)$. Thus for any string $x \in \Sigma^*$.

$$M(x^n) = \alpha_{n-1} M(x^{n-1}) + \cdots + \alpha_1 M(x) + \alpha_0.$$

and for any $i \geq 0$.

$$\begin{aligned} M(x^{n+i}) &= M(x^n)M(x^i) \\ &= \alpha_{n-1}M(x^{n-1})M(x^i) + \cdots + \alpha_1 M(x)M(x^i) + \alpha_0 M(x^i) \\ &= \alpha_{n-1}M(x^{n+i-1}) + \cdots + \alpha_1 M(x^{i+1}) + \alpha_0 M(x^i). \end{aligned}$$

Similarly, for any $y, z \in \Sigma^*$.

$$\begin{aligned} M(yx^{n+i}z) &= M(y)M(x^{n+i})M(z) \\ &= M(y)(\alpha_{n-1}M(x^{n+i-1}))M(z) + \cdots + \\ &\quad M(y)(\alpha_1 M(x^{i+1}))M(z) + M(y)(\alpha_0 M(x^i))M(z) \\ &= \alpha_{n-1}M(yx^{n+i-1}z) + \cdots + \alpha_1 M(yx^{i+1}z) + \alpha_0 M(yx^i z). \end{aligned}$$

Thus

$$f_N(yx^{n+i}z) = \alpha_{n-1}f_N(yx^{n+i-1}z) + \cdots + \alpha_1 f_N(yx^{i+1}z) + \alpha_0 f_N(yx^i z). \tag{3.8}$$

Assume that $yz, yxz, \cdots, yx^{n-1}z \in L$. Let $0 \leq k \leq n-1$, we have

$$f_N(yz) = \cdots f_N(yx^{n-1}z) = \rho. \tag{3.9}$$

By Equation 3.8, we have, for any $n-1 < k \leq m$.

$$\begin{aligned} f_N(yx^n z) &= \alpha_{n-1}f_N(yx^{n-1}z) + \cdots + \alpha_1 f_N(yxz) + \alpha_0 f_N(yz) \\ &= (\alpha_{n-1} + \cdots + \alpha_1 + \alpha_0)\rho \\ &= \rho \qquad (\text{since } \alpha_{n-1} + \cdots + \alpha_1 + \alpha_0 = 1) \\ &\vdots \\ f_N(yx^{n+i}z) &= \alpha_{n-1}f_N(yx^{n+i-1}z) + \cdots + \alpha_1 f_N(yx^{i+1}z) + \alpha_0 f_N(yx^i z) \\ &= (\alpha_{n-1} + \cdots + \alpha_1 + \alpha_0)\rho \\ &= \rho \qquad (\text{since } \alpha_{n-1} + \cdots + \alpha_1 + \alpha_0 = 1). \end{aligned}$$

Therefore, by letting $\rho = \frac{1}{2}$, we have $f_N(yx^m z) = \frac{1}{2}$ for any $m \geq 0$. □

Lemma 3.2.11 $L_{neq} \notin$ **1-C$_=$LIN**.

Proof. The proof starts by assuming, in contrary, that L_{neq} is in **1-C$_=$LIN**. Let $m = k+1$ for any $k \in \mathbb{N}$. We set $x = a$, $y = a$, and $z = b^m$. Then by definition, we have $yz = ab^m$, $yxz = a^2b^m, \cdots, yx^{k-1}z = a^{m-1}b^m \in L_{neq}$. By Theorem 3.2.10, we obtain $yx^k b^m = a^m b^m \in L_{neq}$; hence we have a contradiction. Therefore, $L_{neq} \notin$ **1-C$_=$LIN**. □

The following theorem is an immediate result from Lemma 3.2.11.

Theorem 3.2.12 *There exists a context-free language not belonging to* co-**1-C$_=$LIN**.

As our main theorem in this section, we have

Theorem 3.2.13 *cequalin is not closed under complementation.*

Proof. Since Proposition 3.2.7 shows that $L_{neq} \in$ co-**1-C$_=$LIN**, and by Lemma 3.2.11, $L_{neq} \notin$ **1-C$_=$LIN**; therefore, the proof is completed. □

Chapter 4

Quantum Computation

4.1 Quantum Mechanics

4.1.1 Notations

To distinguish quantum state from classical state, we use Dirac's notation. In Dirac's notation, some symbols are placed inside the "ket" notation $|\ \rangle$ and the "bra" notation $\langle\ |$ to identify an element in an inner-product space V and in V^*, respectively. For a vector $|\psi\rangle$ and a dual vector $\langle\phi|$, the result of applying $|\psi\rangle$ to $\langle\phi|$ is denoted as $\langle\psi|\phi\rangle$, and it is called "bracket". Note that $\langle\psi|\phi\rangle$ is a real scalar, and $|\psi\rangle\langle\phi|$ is a matrix.

Let M be a unitary matrix whose rows and columns are indexed by configurations and $|\phi\rangle$ be a probability distribution at each time step. Then the product $M|\phi\rangle$ represents the distribution of next step. The unitary matrix M is referred as the *time evolution operator*, the distribution at each step is called the *linear superposition* of configuration, and the coefficient of each configuration is the probability known as the *amplitude*.

A *state* in quantum mechanics describes a physical system in some way. A *ray* is an equivalence class of vectors differed by the multiplication of a nonzero complex scalar: $\{ax : a \in \tilde{\mathbb{C}}\}$. A normalized vector is a vector with the norm 1: that is, $\|x\| = 1$. A *quantum state* is a ray in Hilbert space.

4.1.2 Qubits

In contrast to classical bits 0 and 1, a quantum bit (or *qubit* for short) can be visualized as a small vector protruding from the center of a Bloch sphere (see Figure 4.1).

Mathematically, let $\mathcal{H}_n$ denote the Hilbert space of dimension n for each natural number n. A qubit is an element of $\mathcal{H}_2$ that has the unit norm. By choosing the standard basis $\{|0\rangle, |1\rangle\}$, a qubit $|\phi\rangle$ is written as $\alpha_0|0\rangle + \alpha_1|1\rangle$ where $\alpha_0, \alpha_1 \in \tilde{\mathbb{C}}$ satisfying $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Let B_n be the standard basis of $\mathcal{H}_n$. A quantum string is a vector of $\mathcal{H}_n$ with unit length, that is, $\sum_{s\in\{0,1\}^n} \alpha_s|s\rangle$ where $\alpha_s \in \tilde{\mathbb{C}}$ with $\sum_{s\in\{0,1\}^n} |\alpha_s|^2 = 1$. Note that a qubit is a quantum string of length 1.

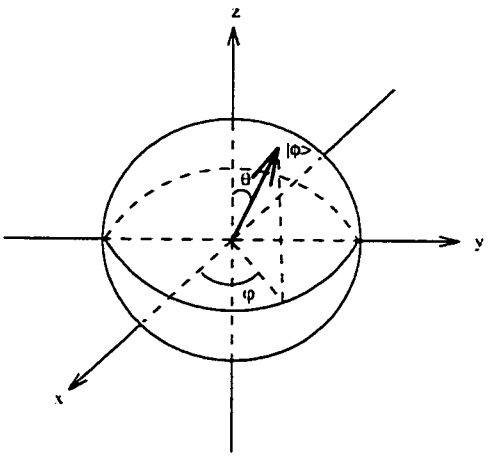

Figure 4.1: A qubit representation in Bloch sphere.

Example 4.1.1

1. The standard basis of a one-qubit system : $\{|0\rangle, |1\rangle\}$.
2. The standard basis of a two-qubit system : $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$.

Some examples of qubit system are expressed as mathematical structures as follows.

Example 4.1.2 (one-qubit system) $|\varphi\rangle = a|0\rangle + b|1\rangle$ where $a, b \in \tilde{\mathbb{C}}$ and $|a|^2 + |b|^2 = 1$.

Example 4.1.3 (two-qubit system) $|\varphi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$ where $a, b, c, d \in \tilde{\mathbb{C}}$ and $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$.

Remark 4.1.4
$$|00\rangle = |0\rangle \otimes |0\rangle$$
$$|10\rangle = |1\rangle \otimes |0\rangle \neq |0\rangle \otimes |1\rangle$$

$$\begin{aligned} |\varphi\rangle &= (a'|0\rangle + b'|1\rangle) \otimes (a''|0\rangle + b''|1\rangle) \\ &= a'a''|0\rangle \otimes |0\rangle + a'b''|0\rangle \otimes |1\rangle + a''b'|1\rangle \otimes |0\rangle + b'b''|1\rangle \otimes |1\rangle \\ &= a'a''|00\rangle + a'b''|01\rangle + a''b'|10\rangle + b'b''|11\rangle. \end{aligned}$$

Example 4.1.5 (k-qubit system)
$|\varphi\rangle = (|\varphi_1\rangle \otimes \cdots \otimes |\varphi_k\rangle)$ where $|\varphi_1\rangle, \cdots, |\varphi_k\rangle$ are single qubits. $|\varphi\rangle = \sum_{|i|=k} a_i|i\rangle$ where $a_i \in \tilde{\mathbb{C}}$ and $\sum_i |a_i|^2 = 1$.

4.1.3 Superposition

A *superposition* of a quantum Turing machine M (or QTM for short) is defined as an element $|o\rangle$ in the inner-product space of finite complex linear combination of configuration of M with

Euclidean norm. The principle of superposition is an ability for quantum systems to exist in a blend of all their allowed states simultaneously rather than existing in just one allowed state at a time. This is impossible in classical implementations.

It is often useful to picture the state of a two-state quantum system as a vector contained in a sphere. The angle that this vector makes with the vertical axis is related to the relative contributions of $|w_0\rangle$ and $|w_1\rangle$ eigenstates to the whole state (see Figure 4.2). Therefore, a qubit is a combination of 0 and 1 such that the vector pointing straight up represents the classical bit 0, and the vector pointing straight down represents the classical bit 1 (see Figure 4.2). The angle that the vector is rotated about the vertical axis corresponds to the "phase." The phase factors do not affect the relative contributions of the eigenstates to the whole state; however, they are crucially important in so-called quantum interference effects. Thus a state can have the some proportions of 0-ness and 1-ness with different amplitudes in accordance to different phase factors. Mathematically, a quantum system can exist in any of n eigenstates $|w_0\rangle, |w_1\rangle, |w_2\rangle, \cdots, |w_{n-1}\rangle$, and it can also exist in the "superposed" state

$$|w\rangle = \sum_{i=0}^{n-1} |w_i\rangle w_i. \tag{4.1}$$

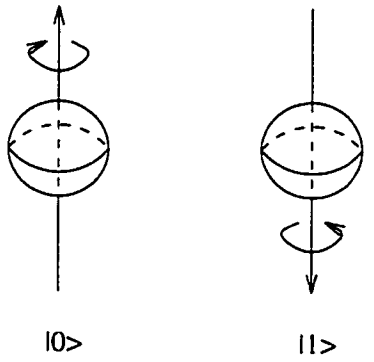

Figure 4.2: Qubit representations of $|0\rangle$ and $|1\rangle$.

4.1.4 Eigenstates

Let $|o_0\rangle$ and $|o_1\rangle$ be two eigenstates of a two-state system. Then the formula $|o\rangle = a|o_0\rangle + b|o_1\rangle$ describes the general state of two-state system as the superposition of these eigenstates. The entire state vector is specified by the two amplitudes a and b of the basis $\{|o_0\rangle, |o_1\rangle\}$. Consider now the standard basis of a one-qubit system: $\{|0\rangle, |1\rangle\}$. In order to represent the binary digit 0, let $a = 1$ and $b = 0$, and to represent the binary 1, let $a = 0$ and $b = 1$. Mathematically, $|o_0\rangle$ and $|o_1\rangle$ can be written as the column vectors $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$, respectively. Thus the binary value 0 is represented as $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, and binary value 1 is represented as $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. The superposition $|o\rangle$ can then be written as:

$$
\begin{aligned}
|\phi\rangle &= a|\phi_0\rangle + b|\phi_1\rangle \\
&= a\begin{pmatrix} 1 \\ 0 \end{pmatrix} + b\begin{pmatrix} 0 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} a \\ b \end{pmatrix}.
\end{aligned}
$$

4.1.5 Measurement

Physically, a qubit can be seen as an electron existing in some space at the same time. We can measure the position of the electron by the light ray and get one point in the space with some probability; however, the measurement destroys the electron state. The similar situation occurs in the quantum state. We can observe the quantum state to get the classical bit 0 or 1 with some probability. After observing a measurement, one cannot simply "undo" the measurement to restore the amplitudes and all states to their original values. In fact, when the original superposition is destroyed, all further measurements on the system will be in accordance with new collapsed superposition. Note that the measurement only gets the classical information instead of quantum information. It can be defffered until the end of computation so that the quantum state remains the same during the computation.

4.2 Models and Definitions

4.2.1 Quantum Turing Machines

A quantum Turing machine is a Turing machine whose read, write, and shift operations are accomplished by quantum mechanical interactions and whose tape content, current state, and head position can exist in a quantum state so-called "superposition" of 0's and 1's, simultaneously. In other words, all possible computations happen at the same time. Such a power is dubbed as "quantum parallelism" and has the potential to encode inputs of a problem in quantum state to perform computations simultaneously on the same tape with one calculation. Unfortunately, we cannot obtain all possible outputs due to the measurement (since quantum state collapses). The final superposed state yields only one output (only classical outcomes can be obtained for human eyes).

There are different ways to understand quantum Turing machines. One is to visualize an exponential number of deterministic Turing machines running simultaneously, each of which happens with some associated amplitude (see Figure 4.3).

The second view (shown in Figure 4.4) of a quantum Turing machine is to see the internal states, the head moves, and the tape contents as vectors in Bloch sphere. Suppose we have a qubit in the finite state control, we can then create a superposition of two states. Similarly, we create superposition with head moves and tape contents.

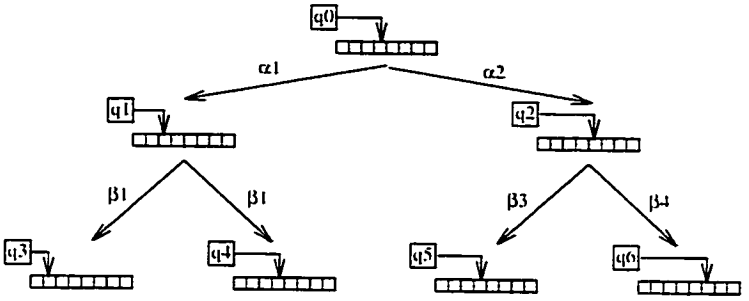

Figure 4.3: Classical view of a quantum Turing machine.

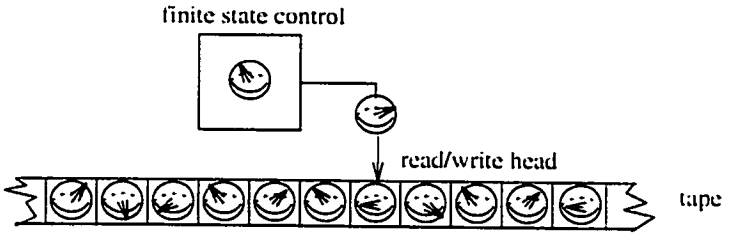

Figure 4.4: Quantum Turing machine from vector space aspect

The formal definition of a one-tape quantum Turing machine is given as follows.

Definition 4.2.1 (one-tape quantum Turing machines) A one-tape quantum Turing machine (or 1QTM for short) M is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F_{acc}, F_{rej})$ where

Q is a finite set of internal states.

Σ is a finite alphabet with a distinguished blank symbol $\{\sqcup\} \notin \Sigma$.

Γ is a tape alphabet with $\Sigma \cup \{\sqcup\} \subseteq \Gamma$.

δ is a transition function: $(Q - F_{acc} - F_{rej}) \times \Sigma \longrightarrow \tilde{\mathbb{C}}^{\Sigma \times Q \times \{L,N,R\}}$.

q_0 is the initial state.

$F_{acc} \subset Q$ is a set of accepting states. and

$F_{rej} \subset Q$ is a set of rejecting states where $F_{rej} \neq F_{acc}$.

4.2.2 A Quantum Analog of co-1-NLIN

Since **NP** is such a popular class, it is understandable that researchers are inspired to explore its quantum analog. It is still controversial whether the quantum analog of **NP** is **NQP** or

QMA. The class **NQP** was first defined by Adleman, Demarrais, and Huang [1] who also showed that **NQP** $\subseteq$ **PP**. It takes the definition of **NP** and substitutes quantum Turing machines for nondeterministic Turing machines with the same accepting criteria. **QMA**, on the other hand, is a quantum version of Arthur-Merlin game. In this section, we take the first definition and focus on the computation of co-**1-NQLIN**, which is a quantum analog of co-**1-NLIN**.

Definition 4.2.2 ($T(n)$-time 1QTM) Let $T:\mathbb{N} \to \mathbb{N}$. A $T(n)$-time 1QTM (abbreviated $T(n)$-1QTM) is a quantum Turing machine M as defined above such that, for each input x, the length of every computation path of M on input x is at most $T(|x|)$.

We define the one-tape linear-time version of co-**NQP** as follows.

Definition 4.2.3 (co-1-NQLIN) Let $T:\mathbb{N} \to \mathbb{N}$. We say that a $T(n)$-1QTM M recognizes a language L if for every x,

$$\begin{aligned} x \in L &\implies \text{Prob}\,[M \text{ accepts } x] = 1, \text{ and} \\ x \notin L &\implies \text{Prob}\,[M \text{ rejects } x] > 0. \end{aligned}$$

The class co-**1-NQTime**($T(n)$) is defined as the set of languages L such that there exists a $T(n)$-1QTM that recognizes L. Then the class co-**1-NQLIN** is defined by

$$\text{co-1-NQLIN} \equiv \text{co-1-NQTime}(O(n)). \tag{4.2}$$

Therefore, co-**1-NQLIN** is the set of languages recognized by one-tape linear-time quantum Turing machines.

Recall that a nondeterministic Turing machine is a ordinary Turing machine with the power of making nondeterministic choice. In the case of co-**1-NQLIN**, even it is called "co-nondeterministic quantum", the power of co-nondeterminism disappears in the implementation of a quantum algorithm. There is no easy way to simulate co-nondeterminism by quantum mechanism. The class co-**1-NQLIN** is categorized by the accepting probability since, after the measurement (quantum state collapses), the result obtained is probabilistic. In addition, the main constraint that ensures a quantum Turing machine to keeps the sum of probabilities equal to 1 is the unitarity of a linear transformation.

4.2.3 Quantum Finite State Automata

A finite state automaton is said to be two-way if the head is allowed to move back and forth along the tape. For simplicity, we assume that end-markers # and \$ indicate, respectively, the leftmost and the rightmost of tape boundaries.

A two-way quantum finite automaton (or 2QFA for short) is defined as follows.

Definition 4.2.4 (2QFA [46]) A 2QFA M is a 6-tuple $(Q, \Sigma, \delta, q_0, F_{acc}, F_{rej})$ where

Q is a finite set of internal states,

Σ is a finite alphabet with $\{\#, \$\} \notin \Sigma$,

δ is a transition function: $Q \times \Sigma \times \{L, N, R\} \longrightarrow \tilde{\mathbb{C}}$.

q_0 is the initial state,

$F_{acc} \subset Q$ is a set of accepting states, and

$F_{rej} \subset Q$ is a set of rejecting states where $F_{acc} \cap F_{rej} = \emptyset$.

Definition 4.2.5 A quantum Turing machine or a quantum finite state automaton is *well-formed* if, at each step, the time evolution matrix is unitary.

4.3 Reversibility

The reversibility of a deterministic computation was first described by Bennett [6]. In retrospect, AND and OR gates are irreversible because of the information loss. The main characteristic of these two gates is that, on given a two-bit information, only a one-bit information is returned. If we can somehow keep the original two-bit information, we can surely reverse the computation. This also implies that if we have only one predecessor (that is, one incoming bit of information) to each successor, we can then reverse the computation.

4.3.1 Reversible Turing Machines

We now define a reversible Turing machine as follows.

Definition 4.3.1 (reversible Turing machines) A one-tape reversible Turing machine (or 1revTM for short) is a 1DTM such that each configuration has at most one predecessor.

Definition 4.3.2 ($T(n)$-time 1revTM) Let $T: \mathbb{N} \to \mathbb{N}$. A $T(n)$-time 1revTM (abbreviated $T(n)$-1revTM) is a reversible Turing machine M such that for each input x, the number of steps of M on input x is at most $T(|x|)$.

Definition 4.3.3 Let $T: \mathbb{N} \to \mathbb{N}$. We say that a 1revTM M is $T(n)$ *time-bounded* if it always halts in either state q_{acc} or q_{rej} at most $T(n)$ steps where n is the size of input. The class 1-rev**DTime**($T(n)$) is defined as the set of languages L such that there exists a $T(n)$-1revTM that recognizes L.

4.3.2 Reversible Computation

To prove the equivalence of **REG** and 1-rev**DTime**($o(n \log n)$), it suffices to show the following lemma.

Lemma 4.3.4 1-**DTime**($o(n \log n)$) = 1-*rev***DTime**($o(n \log n)$).

By definition, it is trivial to see that **1-revDTime**$(o(n \log n)) \subseteq$ **1-DTime**$(o(n \log n))$. The other inclusion is not obvious since our model is restricted to one-tape linear-time computations. Recently, Kondacs and Watrous [46] demonstrated that any DFA can be simulated by some two-way reversible finite automaton (or 2RFA for short). Since any 2RFA is indeed a 1revTM, we can obtain that **1-DTime**$(o(n \log n)) \subseteq$ **1-revDTime**$(o(n \log n))$. For completion purpose, we provide the proposition shown by Kondacs and Watrous.

Proposition 4.3.5 ([46]) *For any DFA M, there exists a 2RFA N such that for any $w \in \Sigma^*$, if M accepts w, N accepts w in $O(|w|)$ steps; and if M does not accept w, N rejects w in $O(|w|)$ steps.*

Proof. In essence, a 2QFA is constructed since a 2RFA is a well-formed 2QFA with amplitudes only 0 and 1. Let L be any language and $M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ be a DFA that recognizes L. First, define a 2DFA $M_2 = (Q_2, \Sigma, \delta_2, q_0', F_2)$ as follows.

$Q_2 = Q_1 \cup \{q_0', q_{acc}', q_{rej}'\}$ where $\{q_0', q_{acc}', q_{rej}'\} \notin Q_1$.

$F_2 = \{q_{acc}', q_{rej}'\}$, and

$$\delta_2(q, \alpha) = \begin{cases} \delta_1(q, \alpha) & q \in Q \wedge \alpha \in \Sigma. \\ q_0 & q = q_0' \wedge \alpha = \#. \\ q_{acc}' & q \in F_1 \wedge \alpha = \$. \\ q_{rej}' & q \in Q_1 - F_1 \wedge \alpha = \$. \text{ and} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

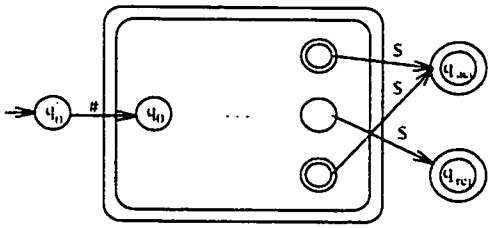

Figure 4.5: DFA to 2DFA.

Let $I_{q,\alpha} = \{p \in Q \mid \delta_2(p, \alpha) = \delta_2(q, \alpha)\}$ be the be the collection of states that has the same next state as q. Let $J_{q,\alpha} = \{p \in Q \mid \delta_2(p, \alpha) = q\}$ be the set of incoming states of q. Assume that all elements in Q_2 are in an increasing order, and *min* and *max* are the minimum and the maximum of such an ordering. For any subset $T \subseteq Q_2$, let $next(q, T)$ be the least element larger than $q \in T$ assuming that such an element exists.

2QFA $N = (Q_N, \Sigma, \delta_N, q_0'', F_{N_{acc}}, F_{N_{rej}})$ is defined as follows:

$Q_N = Q_2 = \{-1, +1\}$.

$q_0'' = (q_0', +1)$.

$F_{N_{acc}} = \{(q'_{acc}, +1)\}$,

$F_{N_{rej}} = \{(q'_{rej}, +1)\}$, and

$$\delta_N(q, \alpha, p, d) = \begin{cases} \langle p|V_\alpha|q\rangle & D(p) = d, \text{ and} \\ 0 & D(p) \neq d \end{cases}$$

where V_α is a linear transformation mapping: $\ell_2(Q) \rightarrow \ell_2(Q)$. D is a mapping: $Q \rightarrow \{-1, 0, +1\}$ such that $D((q, +1)) = +1$ and $D((q, -1)) = -1$, and $\alpha \in \Sigma$.

For each $q \in Q_2$ and $\alpha \in \Sigma$ for which $\delta_2(q, \alpha)$ is defined, let

$$V_\alpha|(q, +1)\rangle = \begin{cases} |(next(q, I_{q,\alpha}), -1)\rangle & p \neq max(I_{q,\alpha}), \text{ and} \\ |(\delta_2(q, \alpha), +1)\rangle & p = max(I_{q,\alpha}). \end{cases}$$

and for each $q \in Q_2$ and $\alpha \in \Sigma$, let

$$V_\alpha|(q, -1)\rangle = \begin{cases} |(q, +1)\rangle & J_{q,\alpha} = \emptyset, \text{ and} \\ |min(J_{q,\alpha}, -1)\rangle & J_{q,\alpha} \neq \emptyset \end{cases}$$

The set of configurations of M_2 on any input w of length n is $Q_2 \times \mathbb{Z}_{n+2}$ (including the symbols # and \$) since, for any finite state automaton, only the current states and the head positions are essential. At each step, the head moves either to left (-1) or right (+1) according to the transition function.

Let G be an undirected graph with set of vertices $Q_2 \times \mathbb{Z}_{n+2}$, and there exists an edge between (q_1, k) and $(q_2, k+1)$ whenever $\delta_2(q_1, w_k) = q_2$. Let G_0 be the connected component of G that contains the initial configuration $(q'_0, 0)$ with no cycle. G_0 contains exactly one vertex to a halting state: that is, either q'_{acc} or q'_{rej}. To reverse the computation, we start at the halting state of G_0, all the leave nodes and all the states that do not have any predecessor. By traversing G_0 in a reversible manner, N can simulate M on input w.

For each configuration (q, α) of M, there are two corresponding configurations of N: $((q, +1), +1)$ and $((q, -1), -1)$. Configuration $((q, +1), +1)$ represents that the subtree rooted at (q, k) has just been traversed, and configuration $((q, -1), -1)$ represents that the subtree rooted at (q, k) is about to be traversed. Let $J_{q,w_k} = I_{q'_i,w_k} = \{q'_1, \cdots q'_m\}$ for each $i = 1, \cdots, m$ shown in Figure 4.6, and assume that $q'_1 < q'_2 < \cdots < q'_{m-1} < q'_m$ according to the ordering of Q_2.

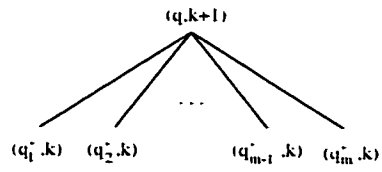

Figure 4.6: A subtree for 2QFA.

Case1: $\mathcal{N}$ is in configuration $((q'_i, +1), k)$ for $i < m$. Since $q'_i \neq max(I_{q'_i, w_k})$, we have, by definition.

$$\begin{aligned} V_\alpha|((q'_i, +1), k)\rangle &= |(next(q'_i, I_{q'_i, w_k}), k-1)\rangle \\ &= |((q'_{i+1}, -1), k-1)\rangle. \end{aligned}$$

Now the subtree rooted at $((q'_{i+1}, -1), k-1)$ is about to be traversed.

Case2: $\mathcal{N}$ is in configuration $((q'_m, +1), k)$. Since $q'_i = max(I_{q'_i, w_k})$, we have, by definition.

$$\begin{aligned} V_\alpha|((q'_m, +1), k)\rangle &= |((\delta_2(q'_m, \alpha), +1), k+1)\rangle \\ &= |((q, +1), k+1)\rangle. \end{aligned}$$

Note that the children of subtree rooted at $((q, +1), k+1)$ have just been traversed.

Case3: $\mathcal{N}$ is in configuration $((q, -1), k)$. Since $J_{q_i, w_k} \neq \emptyset$, we have, by definition.

$$\begin{aligned} V_\alpha|((q, -1), k)\rangle &= |(min(J_{q_i, w_k}), k-1)\rangle \\ &= |((q'_1, -1), k-1)\rangle. \end{aligned}$$

Now the subtree rooted at $((q'_1, -1), k-1)$ is about to be traversed.

Case4: $\mathcal{N}$ is in configuration $((q, -1), k)$ and has no predecessors: that is, $k = 0$ and $q \neq q_0$. Since $J_{q_i, w_k} = \emptyset$, we have, by definition.

$$V_\alpha|((q, -1), k)\rangle = |((q, +1), k+1)\rangle.$$

Since the subtree rooted at $((q, +1), k+1)$ has only one vertex, it has been traversed.

By traversing G_0 in the above manner, $\mathcal{N}$ will eventually reach one of the two configurations $((q'_{acc}, +1), 0)$ or $((q'_{rej}, +1), 0)$ and consequently accepts or rejects, accordingly. It is clear that $\mathcal{N}$ halts after $O(|w|)$ steps, since there are $O(|w|)$ configurations of $\mathcal{N}$, and no configuration may be entered more than once before a halting configuration is reached. This is true of any 2RFA that eventually halts. Since L can be recognized by some DFA, there exists a 2RFA that recognizes L. □

The next proposition follows by the previous lemma and the fact that **1-DTime**$(o(n \log n))$ collapses to **REG**.

Proposition 4.3.6 **REG = 1-*rev*DTime**$(o(n \log n))$.

4.4 Non-regular Sets in co-1-NQLIN

We define a one-tape linear-time quantum Turing machine that recognizes a context-free language using essentially the method shown by Abainis and Watrous [2].

Lemma 4.4.1 $L_{eq\frac{2}{3}} = \{a^i b^j c^k \mid i,j,k \geq 0 \text{ and either } i = j \text{ or } j = k\} \in$ co-1-NQLIN

Proof Sketch. The main idea is to create a superposition of two states using one qubit. Initially, a quantum Turing machine starts with one of the states. At each step, it is rotated with some angle, and meanwhile it creates the superposition.

For each a read, the qubit gets rotated until a b is encountered. For each b read, the machine rotates the qubit (superposed states) in the opposite direction. Clearly, if the number of a's and b's are the same, the initial state should be reached. On the other hand, if the number is different, the machine has to ensure the probability is less than 1. At this instance, we have to chose a irrational number. If a rational number is chosen, it is possible that the initial state would be reached even the numbers of a's and b's are different. We demonstrate an example (see Figure 4.7) by assuming that the angle is $(\pi/4)$. Then the number of a's only needs to be the multiple of the number of b's and vice versa. In addition, the rotation for b's is an exact opposite angle of a's to ensure the numbers for both a's and b's are identical.

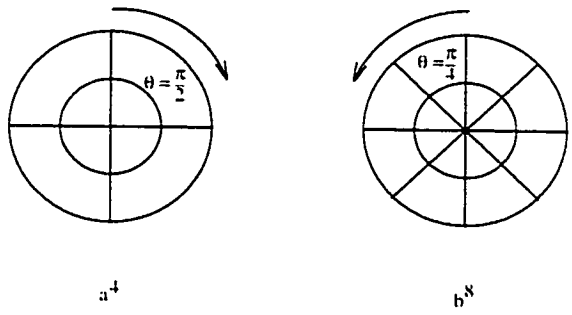

Figure 4.7: An example for rational angles.

Therefore, the angle $\sqrt{2}\pi$ is chosen. Note that it does not work on the arbitrary irrational numbers. The proof depends on the property of $\sqrt{2}$. ■

Proof of Lemma 4.4.1. Let

$$U_\alpha = \begin{pmatrix} \cos(\sqrt{2}\pi) & -\sin(\sqrt{2}\pi) \\ \sin(\sqrt{2}\pi) & \cos(\sqrt{2}\pi) \end{pmatrix} \text{ and } U_{-\alpha} = \begin{pmatrix} \cos(\sqrt{2}\pi) & \sin(\sqrt{2}\pi) \\ -\sin(\sqrt{2}\pi) & \cos(\sqrt{2}\pi) \end{pmatrix}.$$

U_α and $U_{-\alpha}$ are unitary matrices, and one reverses the other since

$$U_\alpha U_{-\alpha} = \begin{pmatrix} \cos(\sqrt{2}\pi) & -\sin(\sqrt{2}\pi) \\ \sin(\sqrt{2}\pi) & \cos(\sqrt{2}\pi) \end{pmatrix} \begin{pmatrix} \cos(\sqrt{2}\pi) & \sin(\sqrt{2}\pi) \\ -\sin(\sqrt{2}\pi) & \cos(\sqrt{2}\pi) \end{pmatrix}$$

$$
\begin{aligned}
&= \begin{pmatrix} \cos^2(\sqrt{2}\pi) + \sin^2(\sqrt{2}\pi) & 0 \\ 0 & \cos^2(\sqrt{2}\pi) + \sin^2(\sqrt{2}\pi) \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
&= I.
\end{aligned}
$$

We define a well-formed 1QTM M that applies the unitary transformations U_α and $U_{-\alpha}$. Each time a transformation is applied, M alternates between state q_0 and state q_1 where q_0 is the initial state. For example, let $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ be the initial-distribution such that the amplitude of q_0 is 1, and the amplitude of q_1 is 0. After applying the first transformation U_α, we obtain

$$
\begin{aligned}
U_\alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} &= \begin{pmatrix} \cos(\sqrt{2}\pi) & -\sin(\sqrt{2}\pi) \\ \sin(\sqrt{2}\pi) & \cos(\sqrt{2}\pi) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} \cos(\sqrt{2}\pi) \\ -\sin(\sqrt{2}\pi) \end{pmatrix}.
\end{aligned}
$$

which changes the amplitude of q_0 from 1 to $\cos(\sqrt{2}\pi)$ and the amplitude of q_1 from 0 to $-\sin(\sqrt{2}\pi)$. M is constructed as follows:

write the input x on the first track and move the head back to the start cell

from left to right, deterministically check if x is of the form $a^i b^j c^k$

if x is not of such a form, reject

else move the head back to the leftmost symbol of a^i (the start cell) and proceed

round1:

(creating the superposition of q_0 and q_1)

start with the leftmost symbol of a^i to the rightmost symbol of a^i, apply U_α and overwrite each a^i by 2

start with the leftmost symbol of b^j to the rightmost symbol of b^j, apply $U_{-\alpha}$ and overwrite each b^j by 1

overwrite each c^k by 0

move the head back to the cell indexed -1

write the quantum state (linear combination of q_0 and q_1)

set the superposed state to q_4 by changing both q_0 and q_1 to q_4.

move the head to the first symbol of 1^j

round2:

(creating the superposition of q_4 and q_5)

start with the leftmost symbol of 1^j to the rightmost symbol of 1^j. apply U_α

start with the leftmost symbol of 0^k to the rightmost symbol of 0^k. apply $U_{-\alpha}$

move the head to the cell indexed -2 (2 cells to the right of the start cell)

write the quantum state (linear combination of q_4 and q_5)

measure both qubits

if q_0 or q_4 is read. write 1; otherwise. write 0

deterministically. check if any of cells indexed -1 and -2 is 1

accept if at least one 1 is read; otherwise. reject.

Since we have to check w in two rounds. we need to ensure that each successor has at most one predecessor. In that case. we overwrite the original a^i. b^j and c^k to distinguish the applications of U_α and $U_{-\alpha}$ between each round.

Remark 4.4.2 Note that for any integer $c \geq 1$.

$$U_\alpha^c = \begin{pmatrix} \cos(c \cdot \sqrt{2}\pi) & -\sin(c \cdot \sqrt{2}\pi) \\ \sin(c \cdot \sqrt{2}\pi) & \cos(c \cdot \sqrt{2}\pi) \end{pmatrix} \text{ and } U_{-\alpha}^c = \begin{pmatrix} \cos(c \cdot \sqrt{2}\pi) & \sin(c \cdot \sqrt{2}\pi) \\ -\sin(c \cdot \sqrt{2}\pi) & \cos(c \cdot \sqrt{2}\pi) \end{pmatrix}.$$

Lemma 4.4.3 *Let p_{rej} be the probability of observing q_1 after n applications of U_α's followed by m applications of $U_{-\alpha}$'s. where $n \neq m$. Then $p_{rej} > \frac{1}{2(n-m)^2}$.*

Proof of Lemma 4.4.3. By the Remark 4.4.2. q_0 gets rotated by $\sqrt{2}(n-m)\pi$. and Q reaches the superposition

$$\cos(\sqrt{2}(n-m)\pi)|q_0\rangle + \sin(\sqrt{2}(n-m)\pi)|q_1\rangle.$$

Then the probability of observing q_1 is $\sin^2(\sqrt{2}(n-m)\pi)$. Let k be the closest integer to $\sqrt{2}(n-m)$. The following two cases are considered:

Case1: If $(\sqrt{2}(n-m) > k)$.

$$\begin{aligned} k^2 &< 2(n-m)^2 \\ &\leq 2(n-m)^2 - 1 \\ k &\leq \sqrt{2(n-m)^2 - 1}. \end{aligned}$$

Therefore,

$$\begin{aligned} & (\sqrt{2}(n-m) - \sqrt{2(n-m)^2 - 1})(\sqrt{2}(n-m) + \sqrt{2(n-m)^2 - 1}) \\ = \; & 2(n-m)^2 - 2(n-m)^2 + 1 \\ = \; & 1. \end{aligned}$$

and

$$\begin{aligned} \sqrt{2}(n-m) - k &\geq \sqrt{2}(n-m) - \sqrt{2(n-m)^2 - 1} \\ &= \frac{1}{\sqrt{2}(n-m) + \sqrt{2(n-m)^2 - 1}} \\ &> \frac{1}{2\sqrt{2}(n-m)}. \end{aligned}$$

Case2: If $(\sqrt{2}(n-m) < k)$.

$$\begin{aligned} k^2 &> 2(n-m)^2 \\ &\geq 2(n-m)^2 + 1 \\ k &\geq \sqrt{2(n-m)^2 + 1}. \end{aligned}$$

Therefore,

$$\begin{aligned} & (\sqrt{2(n-m)^2 + 1} - \sqrt{2}(n-m))(\sqrt{2(n-m)^2 + 1} + \sqrt{2}(n-m)) \\ = \; & 2(n-m)^2 - 2(n-m)^2 + 1 \\ = \; & 1. \end{aligned}$$

and

$$\begin{aligned} k - \sqrt{2}(n-m) &\geq -(\sqrt{2(n-m)^2 + 1} - \sqrt{2}(n-m)) \\ &= -\frac{1}{\sqrt{(2(n-m)^2 + 1} + \sqrt{2}(n-m)} \\ &> -\frac{1}{2\sqrt{2}(n-m)}. \end{aligned}$$

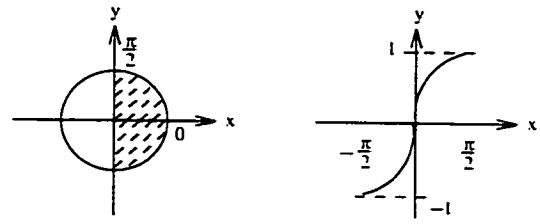

Figure 4.8: Unit circle.

Since k is the closest integer to $\sqrt{2}(n-m)$, we have $-\frac{1}{2} \leq \sqrt{2}(n-m) - k \leq \frac{1}{2}$. We now consider the two intervals (see Figure 4.8): $[-\frac{1}{2}, 0]$ and $[0, \frac{1}{2}]$. For any y in the first interval, $\sin(y\pi) \leq 2y$ since this is true for $y = -\frac{1}{2}$ and $y = 0$, and sin is negative and concave in this interval. Secondly, for any y is the latter interval, $\sin(y\pi) \geq 2y$ since this is true for $y = 0$ and $y = \frac{1}{2}$, and sin is positive and concave in this interval. In both cases, $\sin^2(y\pi) \geq 4y^2$. Thus

$$\begin{aligned}
\sin^2(\sqrt{2}(n-m)\pi) &= \sin^2(\sqrt{2}(n-m) - k)\pi) \\
&\geq 4(\sqrt{2}(n-m) - k)^2 \\
&> \frac{4}{(2\sqrt{2}(n-m))^2} \\
&= \frac{1}{2(n-m)^2}.
\end{aligned}$$

□ Lemma 4.4.3

Case1: $(i = j = k)$ In this case, we have $(U_a^i U_b^j) = (U_a^j U_b^k) = I$ since $i = j = k$. Thus we have $I \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Hence the probability of measuring q_0 is exactly 1.

Case2: $(i = j \wedge j \neq k)$ In this case, we have $(U_a^i U_{-a}^j) = I, (U_a^j U_{-a}^k) = U_a^j U_{-a}^k$ since $i = j$. The accepting probability is 1 since the first qubit is measured with probability 1.

Case3: $(i \neq j \wedge j = k)$ In this case, we have $(U_a^i U_{-a}^j) = U_a^i U_{-a}^j (U_a^j U_{-a}^k) = I$ since $j = k$. The accepting probability is 1 since the last qubit is measured with probability 1.

Case4: $(i \neq j \neq k)$ In this case, we have $(U_a^i U_a^j)(U_a^j U_{-a}^k)$. By Lemma 4.4.3, the rejecting probability is $\frac{1}{2(i-j)^2} + \frac{1}{2(j-k)^2} > 0$.

In conclusion, M accepts x with probability 1 if $x \in L_{eq_{\frac{2}{3}}}$, and M rejects x with probability greater than 0 if $x \notin L_{eq_{\frac{2}{3}}}$. Therefore, $L_{eq_{\frac{2}{3}}} \in$ co-1-**NQLIN**.

□

Lemma 4.4.4 *There exists a non-regular language in* co-1-**NQLIN**.

Proof. The proof for non-regularity of $L_{eq_{\frac{2}{3}}}$ is similar to the proof of Appendix A Lemma A.0.4. Since $L_{eq_{\frac{2}{3}}}$ is not regular, the rest of proof is followed by Lemma 4.4.1. □

Corollary 4.4.5 $L_{eq_3} \in$ co-1-**NQLIN**.

The proof of the above corollary followed by changing the accepting criteria of Lemma 4.4.1 such that M accepts if exactly two 1's are measured.

Lemma 4.4.6 *There exists a non-context-free language in* co-1-**NQLIN**.

Proof. The proof that L_{eq_3} is context-free is shown in Appendix A Lemma A.0.5 The rest of proof is followed by Corollary 4.4.5 since L_{eq_3} is not context-free. □

4.5 Closure Property of co-1-NQLIN

By using the properties of unitary matrices and the characteristics of quantum superposition, we modify the proof of Theorem 3.2.10 and give the following theorem.

Theorem 4.5.1 *Assume that* $L \in$ co-1-**NQLIN**. *Then there exists an* $n \in \mathbb{N}$ *such that for any* $x, y, z \in \Sigma^*$,

$$yz, yxz, \cdots, yx^{n-1}z \in L \implies \forall m \geq 0\ [yx^m z \in L].$$

Proof. The proof is based on the proof for Theorem 3.2.10. Let $L \in$ co-1-**NQLIN**. Then there exists a well-formed 1QTM N such that $L = \{u \in \Sigma^* \mid f_N(u) = 1\}$. Let Q be the set of states for N of cardinality n. Then linear transformation $U(x)$ is an $n \times n$ unitary matrix. Let v be nonzero vector such that $U(x) \cdot v = \lambda \cdot v$ where λ is an eigenvalue of $U(x)$. Then the characteristic function of $U(x)$ has the following form:

$$\begin{aligned} 0 &= 1 \cdot \lambda^n - \alpha_{n-1}\lambda^{n-1} - \cdots - \alpha_1\lambda - \alpha_0, \text{ or} \\ 1 \cdot \lambda^n &= \alpha_{n-1}\lambda^{n-1} + \cdots + \alpha_1\lambda + \alpha_0. \end{aligned}$$

We use the remark from Nasu and Honda [59] that the characteristic polynomial of a stochastic matrix has the following property

$$\alpha_{n-1} + \cdots + \alpha_1 + \alpha_0 = 1.$$

Since each step of N can be interpreted as a linear transformation U_a applied to current quantum state, say $|\varphi\rangle$, and by the property of unitary matrix, the result of $U_b = U_a{}^c$ for some integer c, remains unitary. Thus we have the following characteristic polynomial for U_b

$$1 \cdot \lambda^n = \beta_{n-1}\lambda^{n-1} + \cdots + \beta_1\lambda + \beta_0.$$

By the property of unitary matrix, $U_b^\dagger U_b = I$, we obtain the characteristic polynomial of $U_b^\dagger$:

$$1 \cdot \lambda^n = \beta_{n-1}^* \lambda^{n-1} + \cdots + \beta_1^* \lambda + \beta_0^*$$

such that

$$\beta_{n-1}\beta_{n-1}^* + \cdots + \beta_1 \beta_1^* + \beta_0 \beta_0^* = 1.$$

Recall that each $\beta_i \beta_i^*$ is equal to the probability distribution of each nomial, and their sum is equal to 1. Also

$$U_b(x) \cdot v = \lambda \cdot v \text{ and } U(x^n) = \overbrace{U(x) \cdots U(x)}^{n}.$$

We replace each eigenvalue λ^l by $U_b(x^l)$. Thus for any string $x \in \Sigma^*$,

$$U_b(x^n) = \beta_{n-1} U_b(x^{n-1}) + \cdots + \beta_1 U_b(x) + \beta_0.$$

and, for any $i \geq 0$,

$$\begin{aligned} U_b(x^{n+i}) &= U_b(x^n) U_b(x^i) \\ &= \beta_{n-1} U_b(x^{n-1}) U_b(x^i) + \cdots + \beta_1 U_b(x) U_b(x^i) + \beta_0 U_b(x^i) \\ &= \beta_{n-1} U_b(x^{n+i-1}) + \cdots + \beta_1 U_b(x^{i+1}) + \beta_0 U_b(x^i). \end{aligned}$$

Thus we obtain, for any $x, y, z \in \Sigma^*$,

$$\begin{aligned} U_b(yx^{n+i}z) &= U_b(y) U_b(x^{n+i}) U_b(z) \\ &= U_b(y)(\beta_{n-1} U_b(x^{n+i-1})) U_b(z) + \cdots + \\ &\quad U_b(y)(\beta_1 U_b(x^{i+1})) U_b(z) + U_b(y)(\beta_0 U_b(x^i) U(z) \\ &= \beta_{n-1} U_b(yx^{n+i-1}z) + \cdots + \beta_1 U_b(yx^{i+1}z) + \beta_0 U_b(yx^i z). \end{aligned}$$

and similarly we obtain

$$U_b^\dagger(yx^{n+i}z) = \beta_{n-1}^* U_b^\dagger(yx^{n+i-1}z) + \cdots + \beta_1^* U_b^\dagger(yx^{i+1}z) + \beta_0^* U_b^\dagger(yx^i z).$$

Therefore,

$$f_N(yx^{n+i}z) = \beta_{n-1}\beta_{n-1}^* f_N(yx^{n+i-1}z) + \cdots + \beta_1 \beta_1^* f_N(yx^{i+1}z) + \beta_0 \beta_0^* f_N(yx^i z). \tag{4.3}$$

Assume that $yz, yxz, \cdots, yx^{n-1}z \in L$. Let $0 \leq k \leq n-1$, we have

$$f_N(yz) = \cdots f_N(yx^{n-1}z) = \rho. \tag{4.4}$$

By Equation 4.3. we have for any $n-1 < k \leq m$.

$$\begin{aligned}
f_N(yx^n z) &= \beta_{n-1}\beta^*_{n-1} f_N(yx^{n-1}z) + \cdots + \beta_1\beta^*_1 f_N(yxz) + \beta_0\beta^*_0 f_N(yz) \\
&= (\beta_{n-1}\beta^*_{n-1} + \cdots + \beta_1\beta^*_1 + \beta_0\beta^*_0)\rho \\
&= \rho \qquad (\text{since } \beta_{n-1}\beta^*_{n-1} + \cdots + \beta_1\beta^*_1 + \beta_0\beta^*_0 = 1) \\
&\vdots \\
f_N(yx^{n+i}z) &= \beta_{n-1}\beta^*_{n-1} f_N(yx^{n+i-1}z) + \cdots + \beta_1\beta^*_1 f_N(yx^{i+1}z) + \beta_0\beta^*_0 f_N(yx^i z) \\
&= (\beta_{n-1}\beta^*_{n-1} + \cdots + \beta_1\beta^*_1 + \beta_0\beta^*_0)\rho \\
&= \rho \qquad (\text{since } \beta_{n-1}\beta^*_{n-1} + \cdots + \beta_1\beta^*_1 + \beta_0\beta^*_0 = 1).
\end{aligned}$$

Therefore. by letting $\rho = 1$. we have $f_N(yx^m z) = 1$ for any $m \geq 0$. □

Lemma 4.5.2 $L_{neq} \notin$ co-**1-NQLIN**.

Proof. The proof starts by assuming. in contrary. that L_{neq} is in co-**1-NQLIN**. Let $m = k+1$ for any $k \in \mathbb{N}$. We set $x = a$. $y = a$ and $z = b^m$. Then by definition. we have $yz = ab^m$. $yxz = a^2b^m$. $\ldots$. $yx^{k-1}b^m = a^{m-1}b^m \in L_{neq}$. By Theorem 4.5.1. $yx^k b^m = a^m b^m \in L_{neq}$; hence we have a contradiction. Therefore. $L_{neq} \notin$ co-**1-NQLIN**. □

The following theorem follows Lemma 4.5.2.

Theorem 4.5.3 *There exists a context-free language not belonging to* co-**1-NQLIN**.

Here. we have our main theorem in this section.

Theorem 4.5.4 co-**1-NQLIN** *is not closed under complementation.*

Proof. Similar to Lemma 4.4.1. we can prove $L_{neq} \in$ **1-NQLIN**. The remaining proof follows Lemma 4.5.2 such that $L_{neq} \notin$ co-**1-NQLIN**. □

4.6 1-C$_=$LIN $\subseteq$ co-1-NQLIN

We obtain the following theorem by applying the main idea of Fenner and Green [27]. which used the essentially the techniques shown by Deutsch [19] and Simon.

Theorem 4.6.1 *Let $T(n) = O(n)$ and L be any language in the synchronized* 1-C$_=$**LIN**. *Then there exist a $T(n)$-1QTM N such and a constant c such that for each x of length n. $Pr[N(x)$ accepts$] = \frac{f^2(x)}{2^{c \cdot n}}$.*

Proof. In this proof, we take Definition 3.2.3 using gap function. Let $c \in \mathbb{N}$ and L be a regular language such that, for all x of length n,

$$f(x) = \frac{|\Sigma^{c \cdot n} \cap L(N)| - |\Sigma^{c \cdot n} - L(N)|}{2}$$

where N is a 1QTM that recognizes L. Let $T(n) = O(n)$ and M be a $T(n)$-1DTM that recognizes L such that for all $\langle x, y \rangle$, $\langle x, y \rangle \in L$ if and only if M accepts $\langle x, y \rangle$. Without loss of generality, we replace M with a $T(n)$-1revDTM M'. Since **1-DTime**$(O(n))$ = **1-revDTime**$(O(n))$, M' also recognizes L. Fix an input x of length n, and let $m = c \cdot n$. Let N be a three-track 1QTM such that the first track contains the input string x, the second track contains $m \leq cn$ non-empty cells, and the last track contains one non-empty cell. Let the content of the reserved cells on the second track be y and the content of the reserved cell on the last cell be b. The second track will be the actual working space, and the cell on the last track is the answer bit. The machine N accepts if b is 1; otherwise, it rejects. We denote a possible configuration of $N(x)$ as a basic state $|x, y, b\rangle$ where $|y\rangle$ is a vector of m qubits and $|b\rangle$ is a single qubit. Initially, we have $y = 0^m$, $b = 0$, and the tape head stays at the start cell. Then from left to right, N applies Hadamard transformation

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

to $|y\rangle$ (to create the superposition on the second track) by scanning the tape once.

Remark 4.6.2 Note that

1. $H|0\rangle = \sqrt{2}(|0\rangle + |1\rangle)$, and
2. $H|1\rangle = \sqrt{2}(|0\rangle - |1\rangle)$.

Hence, by applying H to a given string, N can create the superposition of length of input. Furthermore,

1. $H(\sqrt{2}(|0\rangle + |1\rangle)) = |0\rangle$, and
2. $H(\sqrt{2}(|0\rangle - |1\rangle)) = |1\rangle$.

This shows that H is reversible.

Thus we obtain

$$I^{\otimes n} H^{\otimes m} I|x, 0^m, 0\rangle = \frac{1}{2^{m/2}} \sum_{y:|y|=m} |x, y, 0\rangle.$$

N then simulates the reversible deterministic computation of M' on input $\langle x, y \rangle$. Since M' runs in linear time, and every 1revTM is a special case of a well-formed 1QTM, the simulation time is linear. Let $M'(x, y) = b'$. N sets $b = b'$. Thus we have the superposition

$$\frac{1}{2^{m/2}} \sum_{y:|y|=m} |x, y, b'\rangle.$$

Afterwards, N repeatedly performs the scan using Hadamard transformation over $m+1$ qubits. This time, the contents of both y and b are in the scan. N, in fact, simulates all possible computations of M' on y and returns the answer b' according to the calculation on y. This leads N into a new superposition

$$|\psi\rangle = \frac{1}{\sqrt{2}} \frac{1}{2^m} \sum_{y:|y|=m} \sum_{y':|y'|=m} \sum_{b':|b'|=1} (-1)^{y\cdot y' + b\cdot b'} |x, y', b'\rangle.$$

We now project (or measure) $|\psi\rangle$ onto $\langle x, 0^m, 1|$ and obtain

$$\begin{aligned} \langle x, 0^m, 1|\psi\rangle &= \frac{1}{\sqrt{2}} \frac{1}{2^m} \sum_{y:|y|=m} (-1)^{y\cdot 0 + b\cdot 1} \\ &= \frac{1}{\sqrt{2}} \frac{1}{2^m} \sum_{y:|y|=m} (-1)^b \\ &= -\frac{1}{\sqrt{2}} \frac{1}{2^{m-1}} f(x). \end{aligned}$$

Finally, N deterministically checks the $m+1$ bits. If N reads the string $0^m 1$, it accepts; otherwise, it rejects. Therefore, $|x, 0^m, 1\rangle$ is the unique accepting configuration of N, and it has the probability amplitude $-\frac{1}{\sqrt{2}} \frac{1}{2^{m-1}} f(x)$. Therefore, the probability is $\frac{f^2(x)}{2} \frac{1}{2^{2m-2}} = \frac{f^2(x)}{2^{2m}}$. □

Conclusions and Further Discussions

We might need more mathematics to help develop theories of computer science and separate classes. One example is the complementation property of nondeterministic logarithmic space, **NL** = co-**NL**, shown by Immerman [38] and Szelepcsényi [69] in 1988. The proof depends on the logical characterizations of **NL**.

In this thesis, we have shown some applications of a crossing sequence, Kolmogorov complexity, and communication complexity in the study of variants of Turing machines. For example, using the crossing sequence technique, we have shown that **1-NTime**($o(n \log n)$) is closed under complementation, and, in contrast, it is known that **1-NTime**($O(n \log n)$) is not. It would be interesting to show some applications of techniques like diagonalization (a method to separate infinite sets) or query complexity (a complexity that counts only the number of queries or "helps" needed during the computation) since techniques such as a crossing sequence might not be able to separate the classes in the polynomial-time hierarchy.

We have also showed that, in $O(n \log n)$ time, nondeterministic and co-nondeterministic computations are more powerful than deterministic computations. However, we do not know if nondeterminism and co-nondeterminism still posses more computation power over determinism beyond such a time bound. Moreover, it is known that the simulation of a 2DTM by some 1DTM takes quadratic time. By combining our result, **1-NTime**($o(n \log n)$) = co-**1-NTime**($o(n \log n)$) = **REG**, it is reasonable to substitute a 2NTM for a 1NTM and consider what classes would **2-NTime**($o(n \log n)$) and co-**2-NTime**($o(n \log n)$) recognize.

It is known that **PP** is closed under intersection and union. However, it is still open whether **1-PLIN** has the same closure properties since, under the one-tape linear-time restriction, a machine cannot even make a copy of the given input. In the bounded-error probabilistic case, if we substitute a 1QTM for 1PTM, we might possibly increase it computability. Furthermore, we proved that **1-C$_=$LIN** contains non-regular languages and non-context-free languages; yet it is interesting to learn what class it recognizes exactly. The author has shown that **1-C$_=$LIN** is not closed under complementation; however, this does not imply that **C$_=$P** $\neq$ co-**C$_=$P**. Before solving this long open problem, we might consider the one-tape version of it to grasp some idea about their characterizations. Moreover, we might ask if solving this problem implies the complementation property of **NP** since they both use nondeterministic computations.

In Section 4.5, we used the property of unitary matrices and the property of characteristic polynomial of a stochastic matrix, and proved the non-closure property of co-**1-NQLIN** under complementation. Unfortunately, the same strategy does not apply for all classical techniques

due to the characteristics of quantum computations. For example, there is no obvious way to translate and use the crossing sequence technique due to the quantum interference. Moreover, even if researchers proposed a quantum version of a classical technique, other difficulties occur. For example, Berthiaume, van Dam, and Laplante [10], and Vitányi [71] independently proposed the idea of quantum Kolmogorov complexity using different approaches; however, there is no easy way or suitable problems to apply those techniques. Interestingly, we can prove the quadratic lower bound for 1DTM recognizing palindromes using Kolmogorov complexity; however, it does not seem doable in quantum case using quantum Kolmogorov complexity.

Moreover, we have shown that $O(n)$-1QTM can recognize some non-regular languages and some non-context-free languages; however, it is still an open question that such a class contains the set of palindromes, which is also context-free. In addition, we have proven that the synchronized **1-C$_=$LIN** is included in co-**1-NQLIN**, we might further consider the opposite direction. Furthermore, solving the problem whether the synchronized **1-C$_=$LIN** is equal to the generalized **1-C$_=$LIN** would clarify the difference or the similarity of the two classes.

Bibliography

[1] L. M. Adleman. J. DeMarrais. and M. A. Huang. Quantum computability. *SIAM Journal of Computing*. 26. pp.1524–1540. 1997.

[2] A. Ambainis and J. Watrous. Two-way automata with quantum and classical states. To appear in *Theoretical Computer Science*. See also LANL cs.CC/9911009. 1999.

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

[4] Y. Bar-Hillel. M. Perles. and E. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift fur Phonetik Sprachwissenschaft und Kommunikationsforshung*. 14. pp.143–172. 1961.

[5] P. A. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as presented by Turing machines. *Journal of Statistical Physics*. 22(5). pp.563–591. 1980.

[6] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*. 17. pp.525–532. 1973.

[7] C. H. Bennett. E. Bernstein. G. Brassard. and U. Vazirani. Strengths and weakness of quantum computing. *SIAM Journal of Computing*. 26. pp.1510–1523. 1997.

[8] E. Bernstein and U. Vazirani. Quantum Complexity Theory. in *Proceedings of the 25th Annual ACM Symposium on Theory of Computation*. pp.11–20. 1993. See also *SIAM Journal of Computing*. 26. pp.1411–1473. 1997.

[9] A. Berthiaume and G. Brassard. Oracle quantum computing. in *Proceedings of the 7th IEEE Conference on Structure in Complexity Theory*. pp.132–137. 1992.

[10] A. Berthiaume. W. van Dam. and S. Laplante. Quantum Kolmogorov Complexity. LANL quant-ph/0005018. 3 May 2000.

[11] A. K. Chandra. D. Kozen. and L. J. Stockmeyer. Alternation. *JACM*. 28(1) pp.114–133. 1981.

[12] N. Chomsky. On certain formal properties of grammars. *Information and Control*. 2. pp.137–167. 1959.

[13] A. Church. An unsolvable problem in elementary number theory. *American Journal of Mathematics*. 58. pp.345–363. 1936.

[14] A. Cobham. The intrinsic computational difficulty of functions. in *Proceedings of the 1964 Congress for Logic. Mathematics and the Philosophy of Science*. pp.24–30. 1964.

[15] A. Condon. Bounded Error Probabilistic Finite State Automata. in *Handbook of Randomized Computing* (edited by Sanguthevar Rajasekaran. Panos M. Pardalos. John H. Reif. and Jose D.P. Rolim). Kluwer. 2001.

[16] A. Condon. L. Hellerstein. S. Pottle. and A. Wigderson. On the power of finite automata with both nondeterministic and probabilistic states. *SIAM Journal of Computing*. 27. pp.739–762. 1998.

[17] S. A. Cook. The complexity of theorem-proving procedures. in *Proceedings of the 3th Annual ACM Symposium on the Theory of Computing*. pp.151–158. 1971.

[18] K. DeLeeuw. E. Moore. C. Shannon. and N. Shapiro. Computability by probabilistic machines. *Automata Studies*. Princeton University Press. pp.183–212. 1956.

[19] D. Deutsch. Quantum theory. the Church-Turing principle. and the universal quantum computer. in *Proceedings of the Royal Society London*. A. 400. pp.97–117. 1985.

[20] D. Deutsch. Quantum computational networks. in *Proceedings of the Royal Society of London*. A. 425. pp.73–90. 1989.

[21] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. in *Proceedings of the Royal Society of London*. A. 439. pp.439–553. 1992.

[22] P. D. Dieû. On a class of stochastic languages. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*. 17. pp.421–425. 1971.

[23] P. Dúriš and Z. Galil. Two tapes are better than one for nondeterministic machines. *SIAM Journal of Computing*. 13(2). pp.219–227. May 1984.

[24] J. Edmonds. Paths. trees. and flowers. *Canadian Journal of Mathematics*. 17. pp.449–467. 1965.

[25] J. Edmonds. Minimum partition of matroid in independent subsets. *Journal of Research of the National Bureau of Standard*. 69(b). pp.67–72. 1965.

[26] J. R. Evey. Application of pushdown store machines. in *Proceedings 1963 Fall Joint Computer Conference*. AFIPS Press. pp.215–227. 1963.

[27] S. Fenner and F. Green. Determining Acceptance Possibility for a Quantum Computation is Hard for the Polynomial Hierarchy. LANL quant-ph/9812056. 1998.

[28] R. P. Feynman. Simulation physics with computers. *International Journal of Theoretical Physics*. 21. pp.467–488. 1982.

[29] L. Fortnow and J. D. Rogers. Complexity limitations on quantum computation. in *Proceedings of the 13th Conference on Computational Complexity*. pp.202–209. 1998.

[30] R. Freivalds. Probabilistic two-way machines. in *Proceedings of the 10th International Symposium on Mathematical Foundations of Computer Science*. Lecture Notes in Computer Science. 118. pp.33–45. Springer-Verlag. 1981.

[31] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal of Computing*. 6. pp.675–695. 1977.

[32] K. Gödel. *Collected Works Volume I: Publications 1929-1936* (edited by S. Feferman. Stanford University: J. W. Dawson. Jr., Pennsylvania State University: S. C. Kleene. University of Wisconson. Madison (Emeritus): G. H. Moor. Mount Allison University. New Brunswick: R. M. Solovay. University of California. Berkeley: and J. van Heijenoort. Brandeis University). Oxford University Press. 2001.

[33] F. C. Hennie. One-Tape. Off-Line Turing Machine Computations. *Information and Control*. 8. pp.553–578. 1965.

[34] D. Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*. 8. pp.437–479. 1901.

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

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

[37] D. A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*. 257. pp.3–4. 161–190 and 275–303. 1954.

[38] N. Immerman. Nondeterministic Space is Closed Under Complementation. *SIAM Journal on Computing*. 17(5). pp.935–938. 1988.

[39] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*. Plenum Press. pp.85–103. 1972.

[40] R. M. Karp and R. J. Lipton. Some Connections between Nonuniform and Uniform Complexity Classes. *STOC*. pp.302–309. 1980.

[41] S. C. Kleen. Lambda-definability and Recursiveness. Duke Mathematical Journal. 2. pp.340–353. 1936.

[42] S. C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*. Princeton University Press, pp.3–41. 1956.

[43] S. C. Kleene. Origins of recursive function theory. in *the 20th Annual Symposium on Foundations of Computer Science*. IEEE. 29-31. pp.371–382. October 1979.

[44] K. Kobayashi. On the structure of one-tape nondeterministic Turing machine time hierarchy. *Theoretical Computer Science*. 40. pp.175–193. 1985.

[45] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*. 1. pp.1–7. 1965.

[46] A. Kondacs and J. Watrous. On the power of quantum finite state automata. in *Proceeding of the 38th Annual Symposium on Foundations of Computer Science*. pp.66–75. 1997.

[47] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press 1996. ISBN 0-521-56067-5

[48] R. E. Ladner. On the Structure of Polynomial Time Reducibility. *JACM*. 22(1). pp.155–171. 1975.

[49] L. Levin. Universal search problems (in Russian). *Problemy Peredachi Informatsii 9*. 3. pp.115–116. 1973.

[50] M. Li. P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications second edition*. Springer-Verlag. 1997.

[51] J. C. H. Lin. K. Tadaki. and T. Yamakami. "One-Tape Linear-Time Turing Machines". in *Proceedings of the 6th Annual Conference on Quantum Information Theory*. pp.119–124. 2002.

[52] I. I. Macarie. Space-efficient deterministic simulation of probabilistic automata. *SIAM Journal of Computing*. 27. pp.448–465. 1998.

[53] S. R. Mahaney. Sparse Complete Sets of NP: Solution of a Conjecture of Berman and Hartmanis. *JCSS*. 25(2): pp.130–143. 1982.

[54] Y. Matijasevic. Enumerable sets are Diophantine. *Doklady Akademiky Nauk SSSR*. 191. pp.279–282. 1970. English translation: *Soviet Math Doklady*. 11. pp.354–357.

[55] G. H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*. 34(5). pp.1045–1079. 1955.

[56] A. R. Meyer and L. J. Stockmeyer. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. *FOCS*. pp.125–129. 1972.

[57] P. Michel. An NP-complete language accepted in linear time by a one-tape Turing machine. *Theoretical Computer Science*. 85. pp.205–212. 1991.

[58] E. F. Moore. Gedanken experiments on sequential machines. *Automata Studies*. Princeton University Press. pp.129–153. 1956.

[59] M. Nasu and N. Honda. Fuzzy Events Realized by Finite Probabilistic Automata. *Information and Control*. 12. pp.284–303. 1968.

[60] M. A. Nielsen and J. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press. 2000.

[61] M. Ozawa and H. Nishimura. Local transition functions of quantum Turing machines. in *Proceedings of the 4th International Conference on Quantum Communication. Complexity. and Measurement*. 1999.

[62] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley. 1994.

[63] C. H. Papadimitriou and M. Yannakakis. "The complexity of facets (and some facets of complexity)". in *Proceedings of the 24th ACM Symposium on the Theory of Computing*. pp.229–234. 1982: See also *J.CSS* 28. pp.244–259. 1984.

[64] E. L. Post. Finite combinatory process. *J. Symbolic Logic*. 1. pp.103–105. 1937.

[65] M. O. Rabin. Probabilistic automata. *Information and Control*. 6. pp.230–245. 1963.

[66] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*. 3. pp.114–125. 1959.

[67] D. R. Simon. On the power of quantum computation. in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. IEEE Press. pp.116–123. 1994. See also *SIAM Journal of Computing*. 26. pp.1340–1349. 1997.

[68] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company. 1997.

[69] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. Acta Informatica. 26(3). pp.279-284. 1988.

[70] A. M. Turing. On Computable Numbers. with an application to the Entscheidungsproblem. in *Proceedings of the London Mathematical Society*. (2)42. pp.230-265. 1936.

[71] P. M. B. Vitányi. Quantum Kolmogorov Complexity Based on Classical Descriptions. LANL quant-ph/0102108. 1 Oct 2001.

[72] K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*. 23. pp.325–356. 1986.

[73] C. P. Williams and S. H. Clearwater. *Explorations in Quantum Computer*. Springer-Verlag New York, Inc. 1998.

[74] T. Yamakami. A foundation of programming a multi-tape quantum Turing machine. in *Proceedings of the 24th Mathematical Foundations of Computer Science*. Lecture Notes in Computer Science. 1672. pp.430–441. 1999. See also LANL quant-ph/9906084.

[75] T. Yamakami and A. C. Yao. $\mathbf{NQP}_{\mathbf{C}} = \text{co-}\mathbf{C}_{=}\mathbf{P}$. *Information Processing Letter*. See also LANL quant-ph/9812032 and *ECCC Technical Report* TR-98-073.

[76] A. C. Yao. Some complexity questions related to distributed computing. in *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*. pp.209–213. 1979.

[77] A. C. Yao. Quantum circuit complexity. in *Proceedings of the 34th IEEE FOCS*. pp.352–361. 1993.

Appendix A

Definition A.0.3 (pigeonhole principle) Let n be the number of holes and $n+1$ be the number of pigeons. Assume that each pigeon must stay in one of the holes. Then there is at least one hole that has more than one pigeon.

Lemma 1.2.8. *Let L be a language. If L is regular, there exists a number p (called pumping length) such that for any string $w \in L$ of length at least p, w can be divided into $w = xyz$ satisfying the following conditions:*

1. *for each $i \geq 0$, $xy^i z \in L$.*
2. *$|y| > 0$, and*
3. *$|xy| \leq p$.*

□

Proof. Let L be any regular language and $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes L. Let $p = |Q|$. Let w be any string in L of length $n \geq p$. Assume that $w = w_1 \cdots s_n$ where $w_i \in \Sigma$. Let $C = (q_1, \cdots, q_{n+1})$ be an accepting computation of M on input w where $q_{i+1} = \delta(q_i, w_i)$ for all i with $1 \leq i \leq n$. Since $|Q| = p < n+1 = |C|$, by pigeonhole principle, there are at least two states in C being the same state. Let q_i be the first of the pair and q_j be the second where $i \neq j$ and $q_i = q_j$. Note that q_j must occur among first $p+1$ places starting q_1 to q_{n+1}: that is, $j \leq p+1$: otherwise, $|Q| \geq j > p$, and we have a contradiction.

We now define strings $x, y, z \in \Sigma^*$ as follows.

$$x = w_1 w_2 \cdots w_{i-1},$$
$$y = w_i w_{i+1} \cdots w_{j-1}, \text{ and}$$
$$z = w_j w_{j+1} \cdots w_n.$$

Since $w = w_1 \cdots w_n$, $|y| = j - i > 0$, and $|xy| = j - 1 \leq p$, the first three conditions are satisfied. We now prove the forth condition as follows (see Figure A.1).

Let $i \geq 0$. Consider the computation of M on input $xy^i z$. When M reads x, M's state changes from q_1 to q_i. Note that $q_i = \delta(q_{i-1}, w_{i-1})$. If $i = 0$, M starts at state $q_j (= q_i)$ on input z and reaches states q_{n+1}, which is a final (an accepting) state. Thus M accepts xz.

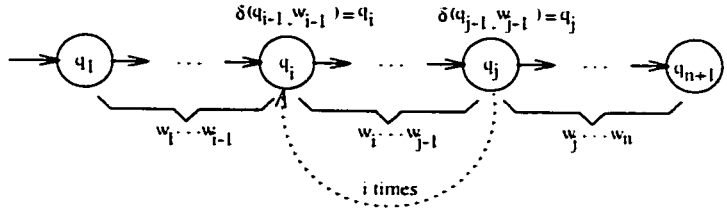

Figure A.1: An example showing xy^iz recognizes by M.

If $i > 0$, M starts at state q_i on input y and reaches state q_j after reading y. Since $q_i = q_j$, M returns its original state q_i after reading y. Now M repeats this procedure i times after reading y^i and reaches state q_j. Obviously, M accepts xy^iz. Hence $xy^iz \in L$. □

Lemma 1.2.13. *Let L be a language. If L is context-free, there exists a number p (called pumping length) such that for any string $w \in L$ of length at least p, w can be divided into $w = uvxyz$ satisfying the following conditions:*

1. *for each $i \geq 0$, $uv^ixy^iz \in L$,*
2. *$|vy| > 0$, and*
3. *$|vxy| \leq p$.*

□

Proof. Let L be any CFL and $G = (V, \Sigma, R, S)$ be a CFG that generates L. Let b be the maximum number of symbols in the body of each rule in R. If $b = 1$, $L(G) \subseteq \Sigma \cap \{\epsilon\}$ since G can only generate strings of length at most 1. If we take $p = 2$, the theorem holds since there is no string of length greater than p.

We now consider the case where $b \geq 2$.

Claim 3 *For any w in L, let h be the maximum height of some parse tree for w. Then $|w| \leq b^h$.*

Proof of Claim 3. The proof is shown by induction on h where $h \geq 1$.

(Basis: $h = 1$) It is trivial since $|w| \leq b$.

(Induction step: $h > 1$) Assume that the claim is true for the height $\leq h - 1$. Consider any parse tree for w. Let $S \rightarrow U_1 \cdots U_m$ be the first rule applied. Note that $m \leq b$. Let w_i be the string generated from U_i by rules such that $w = w_1 \cdots w_m$. By the induction hypothesis, $|w_i| \leq b^h$ for $i = 1, \cdots, m$. Thus $|w| \leq b \cdot b^{h-1} = b^h$.

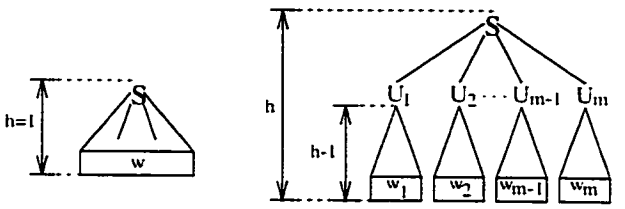

Figure A.2: An example for Claim 3.

□

Let $p = b^{|V|+2}$. Note that $p > b^{|V|+1}$ since $b \geq 2$. Let $w \in L$ be any string of length $n \geq p$. We pump w as follows. Let τ be a parse tree for w that has the minimum number of nodes. Consider the height h of the parse tree τ. Since $b^h \geq |w| \geq p$, by Claim 3, it follows that $h \geq |V| + 2$; otherwise, $|w| \leq b^{|V|+1} < p$, and we have a contradiction. Thus there is a path r in τ that has length greater than $|V| + 2$. Note that r has at least $|V| + 1$ variables (or nonterminal symbols). Since G has $|V|$ variables, by pigeonhole principle, there is a variable T that appears at least twice in r. Assume that T is the variable that repeats among the lowest $|V| + 1$ variables on path r. Next we define u, v, x, y, z as in Figure A.3.

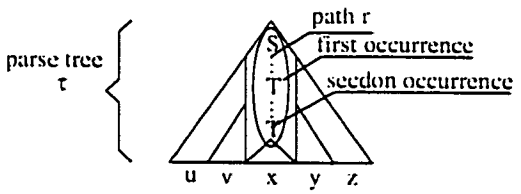

Figure A.3: An example for parse tree τ.

Clearly, the first condition holds since $w = uvxyz$. To prove the second condition, assume, in contrary, that $v = y = \epsilon$. In this case, we substitute the lower occurrence of T for the upper occurrence of T. Let τ' be the new parse tree shown in Figure A.4. Notice that τ' is a parse tree of w that has the number of nodes less than τ. Thus this contradicts the choice of τ that τ has the minimum number of nodes. Therefore, $|vy| > 0$.

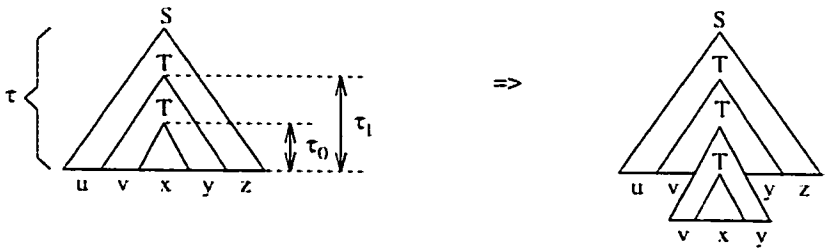

Figure A.4: An example for parse tree τ.

Since r is the longest path, and T is chosen to fall within the bottom $|V|+1$ variables on r. Hence the length of the subtree of the upper occurrence of T that generates vxy is at most $|V|+2$. By Claim 3, $|vxy| \leq b^{|V|+2} = p$. Therefore, the third condition holds.

Finally, we show that $uv^ixy^iz \in L$ for every $i \geq 0$. If $i = 0$, we substitute the lower occurrence of T for the upper occurrence of T (see Figure A.4) and obtain a valid parse tree for uxz. Thus $uxz \in L$. Assume that $i > 0$. Let τ_0, τ_1 be subtrees of the lower occurrence and upper occurrence, respectively in the parse tree τ. We substitute τ_1 for τ_0 and obtain a valid parse tree for uv^2xy^2z. If we repeat this procedure for i times, a valid parse uv^ixy^iz is obtained. Thus $uv^ixy^iz \in L$. □

Lemma A.0.4 *$L_{eq_{\frac{2}{3}}}$ is not regular.*

Proof. We show that L_{eq} is not regular by contradiction. Assume that L_{eq} is regular. By the pumping lemma (for regular language), let p be the pumping length. Consider the string $w = a^pb^p$. Since $|w| > p$, there are x, y, and z where $w = xyz$ that satisfy the pumping lemma: that is,

i. for every $i \geq 0$, $xy^iz \in L_{eq_{\frac{2}{3}}}$.

ii. $|y| > 0$, and

iii. $|xy| \leq p$.

Since $|xy| \leq p$, we have $x = a^m$ and $y = a^k$ for some $m, k \in \mathbb{N}$. Since $|y| > 0$, we have $k > 0$. Thus $z = a^{p-(m+k)}b^p$. Note that, for any $i \geq 0$, $xy^iz = a^m(a^k)^ia^{p-(m+k)}b^p = a^{p-k(i-1)}b^p$. Since $xy^iz \in L_{eq}$, $p + k(i-1) = p$, which implies $k(i-1) = 0$. Since i is arbitrary, we have $k = 0$. This contradicts the fact that $k > 0$. Therefore, L_{eq} is not regular. □

Lemma A.0.5 *L_{eq_3} is not context-free.*

Proof. We assume that L_{eq_3} is context-free. Since L_{eq_3} is a CFL, by the pumping lemma (for context-free language), let p be the pumping length. Consider $w = a^pb^pc^p$. Clearly, w is a member of L_{eq_3} of length at least p. Since $|s| > p$, there are u, v, x, y, and z where $w = uvxyz$, that satisfy the pumping lemma: that is,

i. for every $i \geq 0$, $uv^ixy^iz \in A$.

ii. $|vy| > 0$, and

iii. $|vxy| \leq p$.

Since all a's and b's have fixed order, thus we consider v and y, each with a substring of only either a's or b's. The second condition states that v and y cannot both be empty. The string vxy must be a substring of either a^pb^p, a^pb^p, or a^pb^p. Consider the case that vxy is a

substring of $a^p b^p$. When we pump w twice, we have v^2xy^2, which includes either more p a's or b's. Thus uv^2xy^2 is not in L_{eq3}. For the other cases, we follow the similar argument to obtain contradictions. Therefore, L_{eq3} is not context-free. □