State

Arnold L. Rosenberg

Dept. of Computer Science, University of Massachusetts, Amherst, MA 01003, USA
rsnbrg@cs.umass.edu

Abstract. The notion of state is fundamental to the design and analysis of virtually all computational systems. The Myhill-Nerode Theorem of Finite Automata theory—and the concepts underlying the Theorem—is a source of sophisticated fundamental insights about a large class of state-based systems, both finite-state and infinite-state systems. The Theorem's elegant algebraic characterization of the notion of state often allows one to analyze the behaviors and resource requirements of such systems. This paper reviews the Theorem and illustrates its application to a variety of formal computational systems and problems, ranging from the design of circuits, to the analysis of data structures, to the study of state-based formalisms for machine-learning systems. It is hoped that this survey will awaken many to, and remind others of, the importance of the Theorem and its fundamental insights.

A dedication. I decided to contribute this piece to this volume because Shimon Even is largely—albeit indirectly—responsible for the piece. I learned about Finite Automata Theory and the Myhill-Nerode Theorem in a course taught by Shimon at Harvard during his last year of graduate school and my first. I further learned from associating with Shimon, during a friendship of more than 42 years, a commitment to effective teaching and the importance of defending strongly held positions, even when they run counter to prevailing trends.

1 Introduction

A paean to the Myhill-Nerode Theorem. The notion of state is fundamental to the design and analysis of virtually all computational systems, from the sequential circuits that underlie sophisticated hardware, to the semantic models that enable optimizing compilers, to leading-edge machine-learning concepts, to the models used in discrete-event simulation. Decades of experience with state-based systems have taught that all but the simplest display a level of complexity that makes them hard—conceptually and/or computationally—to design and analyze. One brilliant candle in this gloomy scenario is the Myhill-Nerode Theorem, which supplies a *rigorous, mathematical*, analogue of the following informal characterization of the notion "state."

> The state of a system comprises that fragment of its history that allows it to behave correctly in the future.

O. Goldreich et al. (Eds.): Shimon Even Festschrift, LNCS 3895, pp. 375–398, 2006.

Superficially, it may appear that this definition of "state" is of no greater *operational* significance than is the foundational identification of the number *eight* with the infinitude of sets that contain eight elements. This appearance is illusory. The Myhill-Nerode Theorem turns out to be a conceptual and technical powerhouse when analyzing a surprising range of problems concerning the state-transition systems that occur in so many guises within the field of computation. Indeed, although the Theorem resides most naturally within the theory of Finite Automata—it first appeared in [13]; an earlier, weaker version appeared in [12]; the most accessible presentation appeared in [15]—it has manifold lessons for the analysis of many problems associated with any state-transition system, even those having infinitely many states.

It is my goal to back up the preceding praise for the Myhill-Nerode Theorem by reviewing both the Theorem and a sampler of its applications. In subsequent sections, I review the work of several researchers from the 1960's, whose work on a variety of problems relating to state-transition systems can be viewed as applying the fundamental insights that underlie the Theorem. While the Theorem originated as a cornerstone of the theory of Finite Automata,[1] several of the systems we consider here are quite removed from the standard Finite Automaton model.

A pedagogical ax to grind. Permit me now to step away from technical matters to pedagogical ones. I argue here via case studies that the Myhill-Nerode Theorem, in the insights that it supplies and the formal settings that it suggests, is one of the real gems of the foundational branch of theoretical computer science. To the extent that this evaluation is accurate, it is regrettable that the Theorem, and its algebraic message and insights, have disappeared from virtually all modern introductory texts on "computation theory," despite that fact that all of these begin with a section on finite automata. For illustration, as I was examining texts for my introductory course in this area, I perused [3, 4, 8, 9, 11, 18] and found the Theorem only in the first edition of [4]; its second edition, [3], no longer presents it! While it is not my intention to speculate at length on why the Theorem has been systematically excluded from the aforementioned texts, I suspect that it is due to a narrowing of attitudes over the years/decades about what constitutes the *foundational* branch of theoretical computer science. Whereas earlier attitudes identified "computation theory" with all approaches to a mathematical foundation—as defined in texts by some compendium of loosely related material from the theories of automata, formal languages, computability, and complexity—modern attitudes seem to posit the overriding importance of complexity theory (even while texts continue to include a smattering of material from the three other theories). Thus, understanding the essential nature of computation, as manifest in the resources required to compute various functions, has largely displaced (in the introductory course, at least) the attempt to develop mathematical tools for understanding the structures that underlie the hardware

[1] In my opinion, only the Kleene-Myhill Theorem, which establishes the equivalence between Finite Automata and "Regular Expressions," rivals the Myhill-Nerode Theorem for importance in the theory of Finite Automata.

and software systems that we build and use. I believe that this trend is unsound, both technically and pedagogically. We present embryonic computer scientists with abstract models that we do little to motivate, and we largely deprive them of exposure to foundational material that is likely to be at least as meaningful to them in their professional lives as much of the esoterica that they are exposed to in what for many is their one and only course on "computation theory."

It is incumbent on me to justify my claims about the Myhill-Nerode Theorem—and, thereby, the more general claims I have just made. I do this by presenting the Theorem and a sampler of its applications. I acknowledge freely that my choice of material—as, perhaps, my position—is personal and eccentric. That said, I hope that the reader will at least find this essay provocative.

A roadmap. We begin by introducing, in Section 2, a very general, unstructured, model of state-transition system, that we call the *Online Automaton*. This model is intended to capture those aspects of a state-transition system that are captured by the Myhill-Nerode Theorem and its underlying concepts. We next turn in Section 3 to Finite Automata, and we develop the Myhill-Nerode Theorem (and its proof) in this, its "natural domain." Section 4 presents two applications of the Theorem to finite-state systems. In Section 4.1, we describe how to use the Theorem to prove that a language is not *regular*—i.e., is not acceptable by a Finite Automaton. We further argue in Section 4.1.2 that the proofs of nonregularity that emerge from the technique proposed in all modern texts—which use the so-called Pumping Lemma for regular languages—are never shorter and are seldom as perspicuous as the proofs advocated in Section 4.1. We invoke Occam's Razor[2] to argue for the reinstatement of the Myhill-Nerode Theorem as *the* fundamental technique for proofs of nonregularity. In Section 4.2, we describe how the Theorem supplies the foundation for the fundamental operation of "minimizing" a Finite Automaton, by coalescing states that are "equivalent" with respect to the language that the automaton accepts. Importantly for applications of the theory, such state minimization is purely algorithmic and requires no understanding of what the automaton does. We next leave the "natural domain" of the Theorem and describe three of its conceptual applications to state-transition systems that are not Finite Automata. Our first "indirect" application, in Section 5.1, describes a result from [6] that, informally, applies the Theorem to Online Automata that accept nonregular languages. This result quantitatively sharpens the Theorem's characterization of nonregular languages as those having infinite "memory requirements," by supplying a lower bound on these "requirements." The next study we review, in Section 5.2, lends structure to the infinitely many states of an Online Automaton, by specifying the organization of the memory that the states control. We arrive, thereby, at the notion of a *multi-tape multi-dimensional online Turing Machine*. Now, such models are not in vogue today, largely because they do not faithfully model the structure of digital computers and their peripherals. However, if one views such "machines" as stylized programs that manipulate multiple data structures—a linear "tape" is a linear list, a two-dimensional "tape" is an orthogonal list, etc.—then

[2] *"Entia non sunt multiplicanda praeter necessitatem"* (William of Occam, 14th cent.)

one can use such a model to advantage to prove nontrivial facts about data structures. The result that we adapt from [2] exposes the impact of memory structure on computational efficiency (specifically, time complexity), within the context of a simple data-retrieval problem. Our final "indirect" application of the Theorem, in Section 6, has implications for some of the voluminous work on probabilistic state-transition systems, such as are quite popular within the artificial-intelligence community. We present one of the most striking results from [14]: Even if Finite Automata are modified to make their state transitions probabilistic, the resulting model still accepts only regular sets when the probability that an input is accepted is always bounded away from the threshold required for acceptance. I close in Section 7 with a closing polemic advocating reinstituting the Myhill-Nerode Theorem within our theoretical computer science curricula.

The thread that connects all of the work we survey is the Myhill-Nerode Theorem and its underlying concepts. We hope that we have done justice to this work and that, after reading this piece, the reader will understand—and, hopefully, sympathize with—our claim that the Myhill-Nerode Theorem is a treasure that should be passed on to subsequent generations.

2 Online Automata and Their Languages

Languages. Let Σ be a finite set of (atomic) symbols (or, an **alphabet**). We denote by $\Sigma^\star$ the set of all finite-length strings of elements of Σ—including the *null string* ε, which is the unique string of length 0. A **word over** Σ is any element of $\Sigma^\star$; a **language over** Σ is any subset $L \subseteq \Sigma^\star$.

Equivalence relations on $\Sigma^\star$, specifically, *right-invariant* ones, cast a broad shadow in the theory, hence, in our survey.

An equivalence relation $\equiv$ on $\Sigma^\star$ is **right-invariant** if, for all $z \in \Sigma^\star$, $xz \equiv yz$ whenever $x \equiv y$.

Our particular focus will be on the following specific (right-invariant) equivalence relation on $\Sigma^\star$, which is defined in terms of a given language $L \subseteq \Sigma^\star$.

$$\text{For all } x, y \in \Sigma^\star: \quad [x \equiv_L y] \text{ iff } (\forall z \in \Sigma^\star)[[xz \in L] \Leftrightarrow [yz \in L]]. \tag{1}$$

The following important result is a simple exercise.

Lemma 1 *For all alphabets Σ and all languages $L \subseteq \Sigma^\star$, the equivalence relation $\equiv_L$ is right-invariant.*

Automata. An **online automaton** M is specified as follows:
$M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is a (finite or infinite) set of *states*;
- Σ is a finite *alphabet*;
- δ is the *state-transition function:* $\delta : Q \times \Sigma \longrightarrow Q$;
- q_0 is M's *initial state;* it is the state M is in when you first "switch it on;"

– $F \subseteq Q$ is the set of *final* (or, *accepting*) states; these are the states that specify M's "response to" each input string $x \in \Sigma^\star$.

In order to make the OA model *dynamic* (so that it can "accept" a language), we need to talk about how an OA M responds to strings, not just to single symbols. We therefore *extend* the state-transition function δ to operate on $Q \times \Sigma^\star$, rather than just on $Q \times \Sigma$. It is crucial that our extension truly *extend* δ, i.e., that it agree with δ on strings of length 1 (which can, of course, be viewed as symbols). We call our extended function $\widehat{\delta}$ and define it via the following induction. For all $q \in Q$:

$$\widehat{\delta}(q, \varepsilon) = q$$
$$(\forall \sigma \in \Sigma,\ \forall x \in \Sigma^\star)\ \widehat{\delta}(q, \sigma x) = \widehat{\delta}(\delta(q, \sigma), x).$$

The first equation asserts that M responds only to the stimuli embodied by non-null strings. In the second equation, the unadorned "δ" highlights the fact that $\widehat{\delta}$ is an *extension* of δ.

We can finally define the **language accepted** (or, *recognized*) by M (sometimes called the "behavior" of M):

$$L(M) \stackrel{\text{def}}{=} \{x \in \Sigma^\star \mid \widehat{\delta}(q_0, x) \in F\}.$$

Since it can cause no confusion to "overload" the semantics of δ, we stop embellishing the extended δ with a hat and just write $\delta : Q \times \Sigma^\star \longrightarrow Q$.

In analogy with the equivalence relation $\equiv_L$ of Eq. 1, which is associated with a language L, we associate with each OA M the following equivalence relation on $\Sigma^\star$.

$$\text{For all } x, y \in \Sigma^\star: \qquad [x \equiv_M y] \ \text{ iff } \ [\delta(q_0, x) = \delta(q_0, y)]. \tag{2}$$

The following is an immediate consequence of how we extended the state-transition function δ to $Q \times \Sigma^\star$, in particular, the fact that $\delta(q_0, xz) = \delta(\delta(q_0, x), z)$.

Lemma 2 *For each OA $M = (Q, \Sigma, \delta, q_0, F)$:*
(a) *the equivalence relation $\equiv_M$ is right-invariant;*
(b) $(\forall x, y \in \Sigma^\star)\ [x \equiv_M y]$ *iff* $[x \equiv_{L(M)} y]$.

3 Finite Automata and the Myhill-Nerode Theorem

A **finite automaton** (**FA**, for short) is an OA, $M = (Q, \Sigma, \delta, q_0, F)$, whose state-set Q is finite. A language L is **regular** iff there is an FA M such that $L = L(M)$.

We now prepare for our presentation of the Myhill-Nerode Theorem, which supplies a rigorous mathematical correspondent of the notion of "state." We begin with some basic definitions, facts, and notation. Let $\equiv$ be any equivalence relation on $\Sigma^\star$.

- For each $x \in \Sigma^\star$, the $\equiv$-*class* that x belongs to is $[x]_\equiv \stackrel{\text{def}}{=} \{y \in \Sigma^\star \mid x \equiv y\}$. (When the subject relation $\equiv$ is clear from context, we simplify notation by writing $[x]$ for $[x]_\equiv$.)
- The *classes* of $\equiv$ *partition* $\Sigma^\star$.
- The *index* of $\equiv$ is the number of *classes* that it partitions $\Sigma^\star$ into.

Theorem 3 ([12, 13, 15]). (The Myhill-Nerode Theorem)
The following statements about a language $L \subseteq \Sigma^\star$ are equivalent.

1. *L is regular.*
2. *L is the union of some of the equivalence classes of a right-invariant equivalence relation over $\Sigma^\star$ of finite index.*
3. *The right-invariant equivalence relation, $\equiv_L$ of Eq. 1 has finite index.*

Note. *The earliest version of the Theorem, in [12], uses* congruences*—i.e., equivalence relations that are both right- and left-invariant.*

Proof. We prove the (logical) equivalence of the Theorem's three statements by verifying the three cyclic implications: statement 1 implies statement 2, which implies statement 3, which implies statement 1.

(1) $\Rightarrow$ (2). Say that the language L is regular. There is, then, a FA $M = (Q, \Sigma, \delta, q_0, F)$ such that $L = L(M)$. Then the right-invariant equivalence relation $\equiv_M$ of Eq. 2 clearly has index no greater than $|Q|$. Moreover, L is the union of some of the classes of relation $\equiv_M$:

$$L \;=\; \{x \in \Sigma^\star \mid \delta(q_0, x) \in F\} \;=\; \bigcup_{f \in F} \{x \in \Sigma^\star \mid \delta(q_0, x) = f\}.$$

(2) $\Rightarrow$ (3). We claim that if L is "defined" via some (any) finite-index right-invariant equivalence relation, $\equiv$, on $\Sigma^\star$, in the sense of statement 2, then the specific right-invariant equivalence relation $\equiv_L$ has finite index. We verify the claim by showing that the relation $\equiv$ must *refine* relation $\equiv_L$, in the sense that every equivalence class of $\equiv$ is totally contained in some equivalence class of $\equiv_L$. To see this, consider any strings $x, y \in \Sigma^\star$ such that $x \equiv y$. By right invariance, then, for all $z \in \Sigma^\star$, we have $xz \equiv yz$. Since L is, by assumption, the union of entire classes of relation $\equiv$, we must have

$$[xz \in L] \quad \text{if, and only if,} \quad [yz \in L].$$

We thus have

$$[x \equiv y] \;\Rightarrow\; [x \equiv_L y].$$

Since relation $\equiv$ has only finitely many classes, and since each class of relation $\equiv$ is a subset of some class of relation $\equiv_L$, it follows that relation $\equiv_L$ has finite index.

(3) $\Rightarrow$ (1). Say that L is the union of some of the classes of the finite-index right-invariant equivalence relation $\equiv_L$ on $\Sigma^\star$. Let the distinct classes of $\equiv_L$ be $[x_1]$, $[x_2]$, $\ldots$, $[x_n]$, for some n strings $x_i \in \Sigma^\star$. (Note that, because of the

transitivity of relation $\equiv_L$, we can identify a class uniquely via any one of its constituent strings. This works, of course, for any equivalence relation.) We claim that these classes form the states of an FA $M = (Q, \Sigma, \delta, q_0, F)$ that accepts L. To wit:

1. $Q = \{[x_1], [x_2], \ldots, [x_n]\}$.
   This set is finite because $\equiv_L$ has finite index.
2. For all $x \in \Sigma^\star$ and all $\sigma \in \Sigma$, define $\delta([x], \sigma) = [x\sigma]$.
   The right-invariance of relation $\equiv_L$ guarantees that δ is a well-defined function.
3. $q_0 = [\varepsilon]$.
   M's start state corresponds to its having read nothing.
4. $F = \{[x] \mid x \in L\}$

One verifies by an easy induction that M is a well-defined FA that accepts L.

4 Applying Myhill-Nerode Concepts to FA's

4.1 Proving that Languages Are Nonregular

FA's are very limited in their computing power due to the finiteness of their memories, i.e., of their sets of states. Indeed, the standard way to expose the limitations of FA's—by proving that a language L is not regular—is to establish somehow that the structure of L requires distinguishing among infinitely many mutually distinct situations.

4.1.1. The Continuation Lemma and Fooling Sets. Given the conceptual parsimony and power of Theorem 3, it is not surprising that the Theorem affords one a simple, yet powerful tool for proving that a language is not regular. This tool is encapsulated in the following corollary, which follows immediately from the equivalence of statements (1) and (3) in the Theorem. For reasons that we hope will become suggestive imminently, we refer to the corollary as "The Continuation Lemma." We maintain that the ensuing development should be viewed as *the primary tool* for proving that a language is not regular.

Lemma 4 (The Continuation Lemma)
Let $L \subseteq \Sigma^\star$ be an infinite regular language. Every sufficiently large set of words over Σ contains at least two words x, y such that $x \equiv_L y$.

> The Continuation Lemma has a natural interpretation in terms of FA's, namely, that an FA M has no "memory of the past" other than its current state. Specifically, if strings x and y lead M to the same state (from its initial state)—i.e., if $x \equiv_M y$—then no continuation/extension of the input string will ever allow M to determine which of x and y it actually read.

One applies the Continuation Lemma to the problem of showing that an infinite[3] language $L \subseteq \Sigma^\star$ is not regular by constructing a *fooling set for* L, i.e., an infinite set of words no two of which are equivalent with respect to L. In other words, an infinite set $S \subseteq \Sigma^\star$ is a fooling set for L if for every pair of words $x, y \in S$, there exists a word $z \in \Sigma^\star$ such that precisely one of xz and yz belongs to L.

> This technique has a natural interpretation in terms of FA's. Since any FA M has only finitely many states, any infinite set of words must (by the pigeonhole principle) always contain two, x and y, that are indistinguishable to M, in the sense that $x \equiv_M y$ (so that $x \equiv_{L(M)} y$; cf. Lemma 2). By the FA version of the Continuation Lemma, no continuation z can ever cause M to distinguish between x and y.

We now consider a few sample proofs of the nonregularity of languages, which suggest how direct and simple such proofs can be when they are based on the Continuation Lemma and fooling sets.

Application 1. *The language*[4] $L_1 = \{a^n b^n \mid n \in \mathsf{N}\} \subset \{a, b\}^\star$ *is not regular.*

We claim that the set $S_1 = \{a^k \mid k \in \mathsf{N}\}$ is a fooling set for L_1. To see this, note that, for any distinct words $a^i, a^j \in S_1$, we have $a^i b^i \in L_1$ while $a^j b^i \notin L_1$; hence, $a^i \not\equiv_{L_1} a^j$. By Lemma 4, L_1 is not regular. □

Application 2. *The language* $L_2 = \{a^k \mid k$ *is a perfect square*$\}$ *is not regular.*

This application requires a bit of subtlety. We claim that L_2 is a fooling set for itself! To see this, consider any distinct words $a^{i^2}, a^{j^2} \in L_2$, where $j > i$. On the one hand, $a^{i^2} a^{2i+1} = a^{i^2+2i+1} = a^{(i+1)^2} \in L_2$; on the other hand, $a^{j^2} a^{2i+1} = a^{j^2+2i+1} \notin L_2$, because $j^2 < j^2+2i+1 < (j+1)^2$; hence, $a^{i^2} \not\equiv_{L_2} a^{j^2}$. By Lemma 4, L_2 is not regular. □

Applications 3 and 4. *The language*[5]

$$L_3 = \{x \in \{0,1\}^\star \mid x \text{ reads the same forwards and backwards; symbolically, } x = x^R\}$$

(whose words are often called "palindromes"), and the language

$$L_4 = \{x \in \{0,1\}^\star \mid (\exists y \in \{0,1\}^\star)[x = yy]\}$$

(whose words are often called "squares"), are not regular.

We claim that the set $S_3 = \{10^k1 \mid k \in \mathsf{N}\}$ is a fooling set for both L_3 and L_4. To see this, consider any pair of distinct words, 10^i1 and 10^j1, from S_3. On the one hand, $10^i110^i1 \in L_3 \cap L_4$; on the other hand, $10^j110^i1 \notin L_3 \cup L_4$; hence, $10^i1 \not\equiv_{L_3} 10^j1$, and $10^i1 \not\equiv_{L_4} 10^j1$. By Lemma 4, neither L_3 nor L_4 is regular. □

---

[3] Easily, every finite language is regular; cf. [4].

[4] N denotes the positive integers. a^n denotes a string of n occurrences of string (or symbol) a.

[5] x^R denotes string x written backwards; e.g., $(\sigma_1\sigma_2\cdots\sigma_{n-1}\sigma_n)^R = \sigma_n\sigma_{n-1}\cdots\sigma_2\sigma_1$.

4.1.2. The Pumping Lemma for Regular Languages. Inexplicably to me, most texts shun the proof strategy of Section 4.1.1, in favor of the more cumbersome—or, at least, never less cumbersome—use of the so-called Pumping Lemma for Regular Languages.

The phenomenon of "pumping" that underlies the Pumping Lemma is a characteristic of any finite closed system. Consider, for instance, any finite semigroup[6], $S = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. Since there are only finitely many *distinct* products in any sequence of the form α_{i_1}, $\alpha_{i_1}\alpha_{i_2}$, $\alpha_{i_1}\alpha_{i_2}\alpha_{i_3}$, ..., where each $\alpha_{i_j} \in S$, there must exist two products in the sequence, say $\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}$ and $\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}\alpha_{i_{k+1}}\cdots\alpha_{i_{k+\ell}}$ such that

$$\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k} \;=\; \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}\alpha_{i_{k+1}}\cdots\alpha_{i_{k+\ell}}$$

within the semigroup. By associativity, then, for all $h \in \mathsf{N}$,

$$\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k} \;=\; \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}(\alpha_{i_{k+1}}\cdots\alpha_{i_{k+\ell}})^h,$$

where the power notation implies iterated multiplication within the semigroup.

Within the context of FA's, the phenomenon of "pumping" manifests itself as follows. Focus on an arbitrary FA $M = (Q, \Sigma, \delta, q_0, F)$. Any word $w \in \Sigma^\star$ of length[7] $\ell(w) \geq |Q|$ can be parsed into the form $w = xy$, where $y \neq \varepsilon$,[8] in such a way that $\delta(q_0, x) = \delta(q_0, xy)$. Since M is deterministic—i.e., since δ is a function—for all $h \in \mathsf{N}$,

$$\delta(q_0, x) = \delta(q_0, xy^h), \tag{3}$$

where, as earlier, the power notation implies iterated concatenation. Since the "pumping" depicted in Eq. 3 occurs also with words $w \in \Sigma^\star$ that admit a continuation $z \in \Sigma^\star$ that places them in $L(M)$—i.e., $wz \in L(M)$—we arrive finally at the Pumping Lemma. (Note the implicit invocation of Lemma 4 in our argument.)

Lemma 5 (The Pumping Lemma for Regular Languages)
For every infinite regular language L, there exists an integer $n \in \mathsf{N}$ such that: Every word $w \in L$ of length $\ell(w) \geq n$ can be parsed into the form $w = xyz$, where $\ell(xy) \leq n$ and $\ell(y) > 0$, in such a way that, for all $h \in \mathsf{N}$, $xy^hz \in L$.

The reader should easily see how to use Lemma 5 to prove that sets are not regular. The technique differs from our fooling set/Continuation Lemma technique mainly in the new (and nonintrinsic!) requirement that one of the "fooling" words must be a prefix of the other. I view this extraneous restriction as a sufficient argument not to use Lemma 5 for proofs of nonregularity. However, a common way of using the Lemma actually mandates looking for undesired "pumping" activity, rather than just for a pair of "fooling" words. For instance, a common pumping-based proof of the nonregularity of the language L_1 of Application 1 notes that the "pumped" word y of Lemma 5:

[6] A *semigroup* is a set of elements that are closed under an associative binary multiplication (denoted here by juxtaposition).
[7] $\ell(w)$ denotes the *length* of the string w.
[8] Of course, we could have $x = \varepsilon$.

1. cannot consist solely of a's, or else the block of a's becomes longer than the block of b's;
2. cannot consist solely of b's, or else the block of b's becomes longer than the block of a's;
3. cannot contain both an a and a b, or else the pumped word no longer has the form "a block of a's followed by a block of b's.

Even when one judiciously avoids this three-case argument by invoking the Lemma's length limit on the prefix xy, one is inviting/risking excessive complication by seeking a string that pumps. For instance, when proving the nonregularity of the language L_3 of palindromes, one must cope with the fact that any palindrome *does* pump about its center. (That is, for any palindrome w and any integer ℓ, if one parses w into $w = xyz$, where x and z both have length ℓ, then, indeed, for all $h \in \mathsf{N}$, the word xy^hz is a palindrome.) Note that we are not suggesting that any of the problems we raise is insuperable, only that they unnecessarily complicate the proof process, hence violate Occam's Razor. The danger inherent in using Lemma 5 to prove that a language is not regular is mentioned explicitly in [9]:

> The pumping lemma is difficult for several reasons. Its statement is complicated, and it is easy to go astray in applying it.

We show now that the condition for a language to be regular that is provided in Lemma 5 is *necessary* but not sufficient. This contrasts with the *necessary and sufficient* condition provided by Theorem 3.

Lemma 6 ([20]) *Every string of length* > 4 *in the nonregular language*

$$L_5 = \{uu^Rv \mid u, v \in \{0,1\}^\star;\ \ell(u), \ell(v) \geq 1\}$$

pumps in the sense of Lemma 5.

Proof. We paraphrase from [20]. Each string in L_5 consists of a nonempty even palindrome followed by another nonempty string. Say first that $w = uu^Rv$ and that $\ell(w) \geq 4$. If $\ell(u) = 1$, then we can choose the first character of v as the nonnull "pumping" substring of Lemma 5. (Of course, the "pumped" strings are uninteresting in this case.) Alternatively, if $\ell(u) > 1$, then, since a^k is a palindrome for every $k > 1$, where a is the first character of u, we can let this first letter be the nonnull "pumping" substring of Lemma 5. In either case, the lemma holds.

Notably, the discussion in [20] ends with the following comment.

> For a practical test that exactly characterizes regular languages, see the Myhill-Nerode theorem.

For the record, Theorem 3 provides a simple proof that L_5 is not regular. Let x and y be distinct strings from the infinite language $L = (01)(01)^\star$, with $\ell(y) > \ell(x)$. (Strings in L consist of a sequence of one or more instances of 01.) Easily,

xx^R is an even-length palindrome, hence belongs to L_5 (with $v = \varepsilon$). However, one verifies easily that yx^R does not begin with an even-length palindrome, so that $yx^R \notin L_5$. To wit, if one could write yx^R in the form uu^Rv, then:

- u could not end with a 0, since the "center" substring 00 does not occur in yx^R;
- u could not end with a 1, since the unique occurrence of 11 in yx^R occurs to the right of the center of the string.

By Lemma 4, L_5 is not regular. □

For completeness, we end this section with a version of Lemma 5 that supplies a condition that is both necessary and sufficient for a language to be regular. This version is rather nonperspicuous and a bit cumbersome, hence, is infrequently taught.

Lemma 7 ([5]) (The Necessary-and-Sufficient Pumping Lemma) *A language $L \subseteq \Sigma^\star$ is regular if, and only if, there exists an integer $n \in \mathsf{N}$ such that: Every word $w \in \Sigma^\star$ of length $\ell(w) \geq n$ can be parsed into the form $w = xyz$, where $\ell(y) > 0$, in such a way that, for all $z \in \Sigma^\star$:*

- *if $wz \in L$, then for all $h \in \mathsf{N}$, $xy^hz \in L$;*
- *if $wz \notin L$, then for all $h \in \mathsf{N}$, $xy^hz \notin L$;*

4.2 Minimizing Finite Automata

Theorem 3 and its proof tell us two important things.

1. The notion of "state" underlying the FA model is embodied in the relations $\equiv_M$. More precisely, a state of an FA is a set of input strings that the FA "identifies," because—and so that—any two strings in the set are indistinguishable with respect to the language the FA accepts.
2. The *coarsest*—i.e., smallest-index—equivalence relation that "works" is $\equiv_L$, so that this relation embodies the *smallest* FA that accepts language L.

We can turn the preceding intuition into an algorithm for minimizing the state-set of a given FA. You can look at this algorithm as starting with any given equivalence relation that "defines" L (e.g., with any FA that accepts L) and iteratively "coarsifying" the relation as far as we can, thereby "sneaking" up on the relation $\equiv_L$.

The resulting algorithm for minimizing a FA $M = (Q, \Sigma, \delta, q_0, F)$ essentially computes the following equivalence relation on M's state-set Q. For $p, q \in Q$,

$$[p \equiv_\delta q] \text{ if, and only if } (\forall x \in \Sigma^\star)[[\delta(p, x) \in F] \Leftrightarrow [\delta(q, x) \in F]]$$

This relation says that no input string will allow one to distinguish M's being in state p from M's being in state q. One can, therefore, coalesce states p and q to obtain a smaller FA that accepts $L(M)$. The equivalence classes of $\equiv_\delta$, i.e., the set

$$\{[p]_{\equiv_\delta} \mid p \in Q\}$$

are, therefore, the states of the smallest FA—call it $\widehat{M}$—that accepts $L(M)$. The state-transition function $\widehat{\delta}$ of $\widehat{M}$ is given by

$$\widehat{\delta}([p]_{\equiv_\delta}, \sigma) \;=\; [\delta(p, \sigma)]_{\equiv_\delta}.$$

Finally, the initial state of $\widehat{M}$ is $[q_0]_{\equiv_\delta}$, and the accepting states are $\{[p]_{\equiv_\delta} \mid p \in F\}$. One shows easily that $\widehat{\delta}$ is well defined and that $L(\widehat{M}) = L(M)$.

We simplify our explanation of how to compute $\equiv_\delta$ by describing an example concurrently with our description of the algorithm. We start with a very coarse approximation to $\equiv_\delta$ and iteratively improve the approximation. Fig. 1 presents the FA

$$M = (\{a, b, c, d, f, g, h\}, \{0, 1\}, \delta, a, \{c\})$$

for our example, in tabular form.

M	q	$\delta(q,0)$	$\delta(q,1)$	$q \in F$?
(start state) $\rightarrow$	a	b	f	$\notin F$
	b	g	c	$\notin F$
(final state) $\rightarrow$	c	a	c	$\in F$
	d	c	g	$\notin F$
	e	h	f	$\notin F$
	f	c	g	$\notin F$
	g	g	e	$\notin F$
	h	g	c	$\notin F$

Fig. 1. The FA M that we minimize.

Our initial partition[9] of Q is $Q - F,\ F$, to indicate that the null string ε witnesses the fact that no accepting state is equivalent to any nonaccepting state. This yields the initial partition of M's states:

$$[a, b, d, e, f, g, h]_1,\ [c]_1$$

(The subscript "1" indicates that this is the first discriminatory step). State c, being the unique final state, is not equivalent to any other state.

Inductively, we now look at the current, time-t, partition and try to "break apart" time-t blocks. We do this by feeding pairs of states in the same block single input symbols. If any symbol leads states p and q to different blocks, then, by induction, we have found a string x that discriminates between them. In detail, say that $\delta(p, \sigma) = r$ and $\delta(q, \sigma) = s$. If there is a string x that discriminates between states r and s—by showing them not to be equivalent under $\equiv_\delta$—then

[9] Recalling that partitions and equivalence relations are equivalent notions, we continue to use notation "$[ab \cdots z]$" to denote the set $\{a, b, \ldots, z\}$ viewed as a block of a partition (= equivalence class).

the string σx discriminates between states p and q. In our example, we find that input "0" breaks the big time-1 block, so that we get the "time 1.5" partition

$$[a, b, e, g, h]_{1.5},\ [d, f]_{1.5},\ [c]$$

and input "1" further breaks the block down. We end up with the time-2 partition

$$[a, e]_2,\ [b, h]_2,\ [g]_2,\ [d, f]_2,\ [c]_2$$

Let's see how this happens. First, we find that $\delta(d, 0) = \delta(f, 0) = c \in F$, while $\delta(q, 0) \notin F$ for $q \in \{a, b, e, g, h\}$. This leads to the "time-1.5" partition (since we have thus far used only one of the two input symbols). At this point, input "1" leads states a and e to block $\{d, f\}$, and it leads states b and h to block $\{c\}$; it leaves state g in its present block. We thus end up with the indicated time-2 partition. Further single inputs leave this partition unchanged, so it must be the coarsest partition that preserves $L(M)$.

The preceding sentence invokes the fact that, by a simple induction, if a partition persists under (i.e., is unchanged by) all single inputs, then it persists under all input strings. We claim that such a stable partition embodies the relation $\equiv_M$, hence, by Lemma 2, the relation $\equiv_{L(M)}$. To see this, consider any two states, p and q, that belong to the same block of a partition that persists under all input strings. Stability ensures that, for all $z \in \Sigma^\star$, the states $\delta(p, z)$ and $\delta(q, z)$ belong to the same block of the partition; hence, either both states belong to F or neither does. In other words: If $\delta(q_0, x) = p$ and $\delta(q_0, y) = q$, for $x, y \in \Sigma^\star$, then for all $z \in \Sigma^\star$, either $\{p, q\} \subseteq F$, in which case $\{xz, yz\} \subseteq L(M)$, or $\{p, q\} \subseteq Q - F$, in which case $\{xz, yz\} \subseteq \Sigma^\star - L(M)$. By definition, then, $x \equiv_M y$.

Returning to the algorithm, we have ended up with the FA $\widehat{M}$ of Fig. 2 as the minimum-state version of M.

$\widehat{M}$	q	$\widehat{\delta}(q, 0)$	$\widehat{\delta}(q, 1)$	$q \in F$?
(start state) $\rightarrow$	$[ae]$	$[bh]$	$[df]$	$\notin F$
	$[bh]$	$[g]$	$[c]$	$\notin F$
(final state) $\rightarrow$	$[c]$	$[ae]$	$[c]$	$\in F$
	$[df]$	$[c]$	$[g]$	$\notin F$
	$[g]$	$[g]$	$[ae]$	$\notin F$

Fig. 2. The FA $\widehat{M}$ that minimizes the FA M of Fig.1.

5 Applying Myhill-Nerode Concepts to Non-FA's

We present three applications of the concepts underlying Theorem 3 to state-transition systems other than FA's. Although our primary motivation is to expose interesting applications of Myhill-Nerode-type characterizations of "state,"

for the sake of completeness, we sketch out the derivations of the results that the characterizations lead to. The first two applications involve OA's that are strictly more powerful than FA's.

5.1 Memory Bounds for Online Automata

By Theorem 3, any OA M that accepts a nonregular language must have infinitely many states. We now present a result from [6] that sharpens this statement via an "infinitely-often" lower bound on the number of states an FA $M^{(n)}$ must have in order to correctly mimic M's (word-acceptance) behavior on all words of length $\leq n$ (thereby providing an "order-n approximation" of M). This bound assumes nothing about M other than its accepting a nonregular language. (Indeed, M's state-transition function δ need not even be computable.) In the context of this survey, this result removes Theorem 3 from the confines of the theory of FA's, by adapting it to a broader class of state-transition systems. This adaptation is achieved by converting the *word-relating* equivalence relation $\equiv_M$ to an *automaton-relating* relation that asserts the equivalence of two OA's on all words that are no longer than a chosen parameter.

Let L be a nonregular language, and let M be an OA that accepts L: $L = L(M)$. For any $n \in \mathsf{N}$, an FA $M^{(n)}$ is an *order-n approximate acceptor* of L or, equivalently, an *order-n approximation* of M if

$$\{x \in L(M^{(n)}) \mid \ell(x) \leq n\} \;=\; \{x \in L \mid \ell(x) \leq n\} \;=\; \{x \in L(M) \mid \ell(x) \leq n\}.$$

We denote by $\mathcal{A}_L(n)$ the (obviously monotonic nondecreasing) number of states in the smallest order-n approximate acceptor of L, as a function of n. This quantity can be viewed as a measure of L's "space complexity," in the sense that one needs $\lceil \log_2 \mathcal{A}_L(n) \rceil$ bi-stable devices (say, transistors) in order to implement an order-n approximate acceptor of L in circuitry.

The conceptual framework of Theorem 3 affords one easy access to a nontrivial lower bound on the "infinitely-often" behavior of $\mathcal{A}_L(n)$, *for any* nonregular language L.

Theorem 8 ([6]). *If the language L is nonregular, then, for infinitely many n,*

$$\mathcal{A}_L(n) > \frac{1}{2}n + 1. \tag{4}$$

Proof. Let M_1 and M_2 be OA's. For any $n \in \mathsf{N}$, we say that M_1 and M_2 are *n-equivalent*, denoted $M_1 \equiv_n M_2$, just when

$$\{x \in L(M_1) \mid \ell(x) \leq n\} \;=\; \{x \in L(M_2) \mid \ell(x) \leq n\}.$$

This relation is, thus, a parameterized extension of the relation $\equiv_M$ that is central to Theorem 3.

Our analysis of approximate acceptors of L builds on the following bound on the "degree" of equivalence of pairs of FA's.

Lemma 9 ([10]) *Let M_1 and M_2 be FA's with s_1 and s_2 states, respectively, such that $L(M_1) \neq L(M_2)$. Then $M_1 \not\equiv_{s_1+s_2-2} M_2$.*

Proof (Proof of Lemma 9). We bound from above the number of partition-refinements that suffice for the state-minimization algorithm of Section 4.2 to distinguish the initial states of M_1 and M_2 (which, by hypothesis, are distinguishable).

Since the algorithm is actually a "state-equivalence tester," we can apply it to state-transition systems that are not legal FA's, as long as we are careful to keep final and nonfinal state segregated from one another. We therefore apply the algorithm to the following "disconnected" FA M. Say that, for $i = 1, 2$, $M_i = (Q_i, \Sigma, \delta_i, q_{i,0}, F_i)$, where $Q_1 \cap Q_2 = \emptyset$. Then $M = (Q, \Sigma, \delta, \{q_{1,0}, q_{2,0}\}, F)$, where

- $Q = Q_1 \cup Q_2$
- for $q \in Q$ and $\sigma \in \Sigma$: $\delta(q, \sigma) = \begin{cases} \delta_1(q, \sigma) \text{ if } q \in Q_1 \\ \delta_2(q, \sigma) \text{ if } q \in Q_2 \end{cases}$
- $F = F_1 \cup F_2$.

Now, the fact that $L(M_1) \neq L(M_2)$ implies: (a) that $q_{1,0} \not\equiv_M q_{2,0}$; ($b$) that neither $Q - F$ nor F is empty. How many stages of the algorithm would be required, in the worst case, to distinguish states $q_{1,0}$ and $q_{2,0}$ within M, when the algorithm starts with the initial partition $\{Q - F, F\}$? Well, each stage of the algorithm, save the last, must "split" some block of the partition into two nonempty subblocks. Since one "split," namely, the separation of $Q - F$ from F, occurs before the algorithm starts applying input symbols, and since $|Q| = s_1 + s_2$, the algorithm can proceed for no more than $s_1 + s_2 - 2$ stages; after that many stages, all blocks would be singletons! In other words, if $p \not\equiv_M q$, for states $p, q \in Q$, then there is a string of length $\leq s_1 + s_2 - 2$ that witnesses the nonequivalence. Since we know that $q_{1,0} \not\equiv_M q_{2,0}$, this completes the proof.

Back to the theorem. For each $k \in \mathsf{N}$, Theorem 3 guarantees that there is a smallest integer $n > k$ such that $\mathcal{A}_L(k) = \mathcal{A}_L(n-1) < \mathcal{A}_L(n)$. The preceding inequality implies the existence of FA's M_1 and M_2 such that:

1. M_1 has $\mathcal{A}_L(n-1)$ states and is an $(n-1)$-approximate acceptor of L;
2. M_2 has $\mathcal{A}_L(n)$ and is an n-approximate acceptor of L.

By statement 1, $M_1 \equiv_{n-1} M_2$; by statements 1 and 2, $M_1 \not\equiv_n M_2$. By Lemma 9, then, $M_1 \not\equiv_{\mathcal{A}_L(n-1)+\mathcal{A}_L(n)-2} M_2$. Since $M_1 \equiv_{n-1} M_2$, we therefore have $\mathcal{A}_L(n-1) + \mathcal{A}_L(n) > n + 1$, which yields Ineq. 4, since $\mathcal{A}_L(n-1) \leq \mathcal{A}_L(n) - 1$.

It is shown in [6] that Theorem 8 is as strong as possible, in that: the constants $\frac{1}{2}$ and 1 in Ineq. 4 cannot be improved; the phrase "infinitely many" cannot be strengthened to "all but finitely many."

5.2 Online Automata with Structured States

The preceding section derives a lower bound on the size of any OA M that accepts a nonregular language L, by bounding the number of classes of $\equiv_L$. In this section, we present lower bounds from [2] on the *time* a specific genre of OA requires to accept a language L, based on the "structure" of the OA's infinitely many states. Specifically, we analyze the behavior of "online" Turing Machines (TM's) whose infinitely many states arise from a collection of read-write "work tapes" of unbounded capacities. As in Section 5.1, the desired bound is achieved by adapting Theorem 3 to a broad class of infinite-state OA's. This adaptation is achieved here by parameterizing the word-relating equivalence relation $\equiv_M$; for each integer $t > 0$, the parameter-t relation $\equiv_M^{(t)}$ behaves like $\equiv_M$, but exposes only discriminations that M can make in t or fewer steps.

A word about TM's is in order, to explain why the study in this section is relevant to computer scientists. The TM model originated in the monumental study [19] that planted the seeds of computability theory, hence, also, of complexity theory. Lacking real digital computers as exemplars of the genre, Turing devised a model that served his purposes but that would be hard to justify today as a way for thinking about either computers or algorithms. Seen in this light, one surmises that TM's persists in today's textbooks on computation theory only because of their mathematical simplicity. However, I believe there is an alternative role for the TM model, which justifies continued attention—in certain contexts. Specifically, one can often devise varieties of TM that allow one to expose the impact of data-structure topology on the efficiency of certain computations. These TM's abstract the control portion of an algorithm down to a finite state-transition system and use the TM's "tapes" to model access to data structures. The study in [2] uses TM's in this way, focusing on the impact of tape topology on efficiency of retrieving sets of words. As such, the bounds here can be viewed as an early contribution to the theory of data structures. This perspective underlay both my "data graph" model [16] and Schönhage's "storage modification machine" model [17]. The interesting features here are the formulation of an information-retrieval problem as a formal language, and the use of the concepts underlying Theorem 3 to analyze the problem.

5.2.1. The Online TM Model. A *d-dimensional tape* is a linked data structure with an array-like topology, termed an *orthogonal list* in [7]. A tape is accessed via a *read-write head*—the TM-oriented name for a pointer. Each cell of a tape holds one symbol from a finite set Γ that contains a designated "blank" symbol; e.g., in a 32-bit computer, Γ could be the set of 32-bit binary words, and the "blank" symbol could be the word of all 1's. Access to cells within a tape is sequential: one can move the head at most one cell in any of the $2d$ permissible directions in a step.

An **online TM** M with t d-dimensional "work tapes" can be viewed as an FA that has access to t d-dimensional tapes. As with any FA, M has an *input port* via which it scans symbols from its input alphabet Σ; it also has a designated initial state and a designated set of final states.

> Let me explain the role of the input port in M's "online" computing, by analogy with FA's. One can view an FA as a device that is passive until a symbol $\sigma \in \Sigma$ is "dropped into" its input port. If the FA is in a stable configuration at that moment—meaning that all bi-stable devices in its circuitry have stabilized—then the FA responds to input σ. The most interesting aspect of this response is that the FA indicates whether the entire sequence of input symbols that it has been presented up to that point—i.e., up to and including the last instance of symbol σ—is accepted. Note that the FA responds to input symbols in an *online* manner, making acceptance/rejection decisions about each prefix of the input string as that prefix has been read. Of course, once the FA has "digested" the last instance of symbol σ, by again reaching a stable configuration, then it is ready to "digest" another input symbol, when and if one is "dropped into" its input port.

The TM M uses its input port in much the manner just described. There is, however, a fundamental difference between an FA and an online TM. During the "passive" periods in which an FA does not accept new input symbols at its input port, the FA is typically waiting for its logic to stabilize, hence is usually not considered to be doing valuable computation. In contrast, during the "passive" periods in which an online TM does not accept new input symbols at its input port, the TM may be doing quite valuable subcomputations using its work tapes. Indeed, the study in [2] can be viewed as bounding (from below) the cumulative time that must be devoted to these "introspective" subcomputations when performing certain computations. With this intuitive background in place, a computational step by M depends on:

- its current state,
- the current input symbol, *if M's program reads the input at this step*,
- the t symbols (elements of Γ) currently scanned by the pointers on the t tapes.

On the basis of these, M:

- enters a new state (which may be the same as the current one),
- independently rewrites the symbols currently scanned on the t tapes (possibly with the same symbol as the current one),
- independently moves the read-write head on each tape at most[10] one square in one of the $2d$ allowable directions.

Notes. **(a)** When $d = 1$, we have a TM with t linear (i.e., one-dimensional) tapes. **(b)** Our tapes have array-like topologies because of the focus in [2]. It is easy to specify tapes with other regular topologies, such as trees of various arities.

One extends M's one-step computation to a multistep computation (whose goal is language recognition, as usual) as follows. To determine if a word $w =$

[10] "At most" means that a read-write head is allowed to remain stationary.

$\sigma_1\sigma_2\cdots\sigma_n \in \Sigma^\star$ is accepted by M—i.e., is in the language $L(M)$—one makes w's n symbols available, in sequence, at M's input port. If M starts in its initial state with all cells of all tapes containing the "blank" symbol, and it proceeds through a sequence of N steps that:

- includes n steps during which M "reads" an input symbol,
- ends with a step in which M is programmed to "read" an input symbol,

then M is said to *decide w in N steps*; if, moreover, M's state at step N is an accepting state, then M is said to *accept w in N steps*.

The somewhat complicated double condition for acceptance ("includes ..." and "ends with ...") ensures that, if M accepts w, then it does so unambiguously. Specifically, after reading the last symbol of w, M does not "give its answer" until it prepares to read the next input symbol (if that ever happens). This means that M cannot oscillate between accepting and nonaccepting states after reading the last symbol of w.

5.2.2. The Impact of Tape Structure on Memory Locality. The *configuration* of an online TM M having t d-dimensional tapes, at any step of a computation, is the $(t+1)$-tuple $\langle q, \tau_1, \tau_2, \ldots, \tau_t\rangle$ defined as follows.

- q is the state of M's finite-state control (its associated FA);
- each τ_i is the d-dimensional configuration of symbols from Γ that comprises the non-"blank" portion of tape i, with one symbol highlighted (in some way) to indicate the current position of M's read-write head on tape i.

(M's configuration is often called its "total state.") The importance of this concept resides in the following. Say that, for $i = 1, 2$, the database-string $x_i \in \Sigma^\star$ leads M to configuration $C_M(x_i) = \langle q_i, \tau_{i1}, \tau_{i2}, \ldots, \tau_{it}\rangle$. If:

- $q_1 = q_2$; i.e., the configurations share the same state;
- for some integer $r \geq 1$, and all $i \in \{1, 2, \ldots, t\}$, tape configurations τ_{1i} and τ_{2i} are identical within r symbols of their highlighted symbols (which indicate where M's read-write heads reside),

then we say that the databases specified by x_1 and x_2 are *r-indistinguishable* by M, denoted $x_1 \equiv_M^{(r)} x_2$. This relation is an important parameterization of the FA-oriented relation $\equiv_M$ that is central to Theorem 3. Specifically, by analyzing relation $\equiv_M^{(r)}$, one can sometimes bound the time-complexity of various subcomputations by M, in the following sense.

Lemma 10 *Say that $x_1 \equiv_M^{(r)} x_2$. If there exists a $y \in \Sigma^\star$ such that one of x_1y, x_2y belongs to $L(M)$, while the other does not, then, having read either of x_1 or x_2, M must compute for more than r steps while reading y.*

5.2.3. An Information-Retrieval Problem Formulated as a Language. The following problem is used in [2] to expose the potential effect of tape structure on computational efficiency. We feed an online TM M a set of equal-length binary words, which we term a *database*. We then feed M a sequence of binary words, each of which is termed a *query*. After reading each query, M must respond "YES" if the query word occurs in the database, and "NO" if not.

The *database language* $L_{DB} \subseteq \Sigma^\star$, where, $\Sigma = \{0, 1, :\}$, and ":" is a symbol distinct from "0" and "1," formalizes the preceding problem. Each word in L_{DB} has the form

$$\xi_1 : \xi_2 : \cdots : \xi_m :: \eta_1 : \eta_2 : \cdots : \eta_n$$

where, for some $k \in \mathsf{N}$,

- each ξ_i $(1 \leq i \leq m)$ and each η_j $(1 \leq j \leq n)$ is a length-k binary string;
- $m = 2^k$;
- $\eta_n \in \{\xi_1, \xi_2, \ldots, \xi_m\}$.

Both the sequence of ξ_i's and the sequence of η_j's can contain repetitions. In particular, we are interested only in the *set* of words $\{\xi_1, \xi_2, \ldots, \xi_m\}$ (the database). The *database string* "$\xi_1 : \xi_2 : \cdots : \xi_m$" is just the mechanism we use to present the database to M. Each word η_j is a *query*. In each word $x \in L_{DB}$, the double colon "::" separates the database from the queries, while the single colon ":" separates consecutive binary words.

The fact that we are interested only in whether or not *the last* query appears in the database reflects the *online* nature of the computation: M must respond to each query as it appears, with no knowledge of which is the last, hence, the important one. (This is essentially the challenge faced by all online algorithms.)

5.2.4. Tape Dimensionality and the Time to Recognize L_{DB}. For simplicity, we focus henceforth on the sublanguages of L_{DB} that are parameterized by the common lengths of their binary words. For each $k \in \mathsf{N}$, $L_{DB}^{(k)}$ denotes the sublanguage in which each ξ_i and each η_j has length k. Note that each database-string in $L_{DB}^{(k)}$ has length $(k+1)2^k - 1$.

Focus on any fixed $L_{DB}^{(k)}$. If the database-strings x_1 and x_2 specify distinct databases, then there exists a query η that appears in the database specified by one of the x_i but not the other—so, precisely one of $x_1 :: \eta$ and $x_2 :: \eta$ belongs to $L_{DB}^{(k)}$. Database-strings that specify distinct databases must, thus, lead M to distinct configurations.

How "big" must these configurations be? On the one hand, there are $2^{2^k} - 1$ distinct databases (corresponding to each nonempty set of length-k ξ_i's). On the other hand, for any M with t d-dimensional tapes, there is an $\alpha_M > 0$ that depends only on M's structure, such that M has $\leq \alpha_M^{dtr}$ distinct configurations of "radius" r—meaning that all non-"blank" symbols on all tapes reside within r cells of the read-write heads. Thus, in order for each database to get a distinct configuration (so that $\equiv_M^{(r)}$ has $\geq 2^{2^k} - 1$ equivalence classes), the "radius" r must exceed $\beta_M \cdot 2^{k/d}$, for some $\beta_M > 0$ that depends only on M's structure.

Combining this bound with Lemma 10, we arrive at the following time bound.

Lemma 11 *If $L(M) = L_{DB}^{(k)}$, then, for some length-k query η, M must take*[11] $> \beta_M \cdot (2^{1/d})^k$ *steps while reading η, for some $\beta_M > 0$ that depends only on M's structure.*

The reasoning behind Lemma 11 is *information theoretic,* depending only on the fact that the number of databases in $L_{DB}^{(k)}$ is doubly exponential in k, while the number of bounded-"radius" TM configurations is singly exponential. Therefore, no matter how M reorganizes its tape contents while responding to one bad query, there must be a query that is bad for the new configuration! By focusing on strings with 2^k bad queries, we thus obtain:

Theorem 12 ([2]). *Any online TM M with d-dimensional tapes that recognizes the language L_{DB} must, for infinitely many N, take time $> \beta_M \cdot (N/\log N)^{1+1/d}$ to process inputs of length N, for some constant $\beta_M > 0$ that depends only on M's structure.*

One finds in [2] a companion upper bound of $O(N^{1+1/d})$ for the problem of recognizing L_{DB}. Hence, Theorem 12 does expose the potential of nontrivial impact of data-structure topology on computational efficiency. In its time, the theorem also exposed one of the earliest examples of the cost of the *online* requirement. Specifically, L_{DB} can clearly be accepted *in linear time* by a TM M that has just a single, linear work tape, but that operates in an *offline* manner—meaning that M gets to see the entire input string before it must give an answer (so that it knows which query is important before it starts computing).

6 Finite Automata with Probabilistic Transitions

We now consider a rather different genre of OA's, namely, FA's whose state-transitions are *probabilistic*, with acceptance decisions depending on the probability of ending up in an final state. This is a very timely model to consider since probabilistic state-transition systems are currently quite in vogue in several areas of artificial intelligence, notably the growing area of machine learning. The main result that we present comes from [14]; it exhibits a nontrivial, somewhat surprising situation in which probabilistic state-transitions add no power to the model: The restricted automata accept only regular sets.

6.1 PFA's and Their Languages

We start with an FA, $M = (Q, \Sigma, \delta, q_0, F)$, and make its state-transitions and acceptance criterion *probabilistic*. We call the resulting model a *Probabilistic Finite Automaton* (*PFA*, for short).

[11] We write $2^{k/d}$ in the unusual form $(2^{1/d})^k$ to emphasize that the dimensionality of M's tapes (which is a fixed constant) appears only in the base of the exponential.

States. We simplify the formal development by positing that the state-set of the PFA M is $Q = \{1, 2, \ldots, n\}$, with $q_0 = 1$, and $F = \{m, m+1, \ldots, n\}$ for some $m \in Q$.

State-transitions. We replace M's state-transition function δ with a set of tables, one for each symbol of Σ. The table associated with $\sigma \in \Sigma$ indicates, for each pair of states $q, q' \in Q$, the probability—call it $\rho_{q,q'}$—that M ends up in state q' when started in state q and "fed" input symbol σ. It is convenient to present the state-transition tables as matrices. The table associated with $\sigma \in \Sigma$ is:

$$\Delta_\sigma \;=\; \begin{pmatrix} \rho_{1,1} & \rho_{1,2} & \cdots & \rho_{1,n} \\ \rho_{2,1} & \rho_{2,2} & \cdots & \rho_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{n,1} & \rho_{n,2} & \cdots & \rho_{n,n} \end{pmatrix}$$

where each[12] $\rho_{i,j} \in [0, 1]$, and, for each i, $\sum_j \rho_{i,j} = 1$.

States, revisited. The probabilistic nature of M's state-transitions forces us to distinguish between M's set of states—the set Q—and the "state" that reflects M situation at any point of a computation, which is a probability distribution over Q. We therefore define the *state-distribution* of M to be a vector of probabilities $\boldsymbol{q} = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$, where each π_i is the probability that M is in state i. The *initial state-distribution* is $\boldsymbol{q}_0 = \langle 1, 0, \ldots, 0 \rangle$.

State-transitions, revisited. Under the preceding formalism, the PFA analogue of the FA single-symbol state-transition $\delta(q, \sigma)$ is the vector-matrix product: $\widehat{\Delta}(\boldsymbol{q}, \sigma) \;=\; \boldsymbol{q} \times \Delta_\sigma$. By extension, the PFA analogue of the FA string state-transition $\delta(q, \sigma_1\sigma_2\cdots\sigma_k)$, where each $\sigma_i \in \Sigma$, is

$$\widehat{\Delta}(\boldsymbol{q}, \sigma_1\sigma_2\cdots\sigma_k) \;\stackrel{\text{def}}{=}\; \boldsymbol{q} \times \Delta_{\sigma_1} \times \Delta_{\sigma_2} \times \cdots \times \Delta_{\sigma_n}. \tag{5}$$

The language accepted by a PFA. The probabilistic analogue of acceptance by final state builds on the notion of an *(acceptance) threshold* $\theta \in [0, 1]$. The string $x \in \Sigma^\star$ is *accepted* by M iff

$$p_M(x) \;\stackrel{\text{def}}{=}\; \sum_{i=m}^{n} \widehat{\Delta}(\boldsymbol{q}_0, x)_i \;>\; \theta,$$

where $\widehat{\Delta}(\boldsymbol{q}, x)_i$ denotes the ith coordinate of the tuple $\widehat{\Delta}(\boldsymbol{q}, x)$. (Recall that M's final states are those whose integer-names are $\geq m$.) Thus, x is accepted iff it leads M from its initial state to its set of final states with probability $> \theta$. As with all OA's, the *language accepted by* M is the set of all strings that M accepts. Acknowledging the crucial role of the acceptance threshold θ, we denote this language by

$$L(M, \theta) \;\stackrel{\text{def}}{=}\; \{x \in \Sigma^\star \mid p_M(x) > \theta\}.$$

[12] As usual, $[0, 1]$ denotes the closed real interval $\{x \mid 0 \leq x \leq 1\}$.

6.2 $L(M, \theta)$ Is Regular when θ Is "Isolated"

It is noted in [14] that even simple—e.g., two-state—PFA's can accept nonregular languages, when accompanied by an "unfavorable" acceptance threshold. When thresholds are "favorable," though, all PFA's accept regular languages.

The threshold $\theta \in [0,1]$ is *isolated* for the PFA M iff there exist a real *constant of isolation* $\varepsilon > 0$ such that, for all $x \in \Sigma^\star$, $|p_M(x) - \theta| \ \geq \ \varepsilon$.

Theorem 13 ([14]). *For any PFA M and associated* isolated *acceptance threshold θ, the language $L(M, \theta)$ is regular.*

Proof. We sketch the proof from [14], which is a direct application of Theorem 3. Say that M has n states, a of which are final, and let $\varepsilon > 0$ be the constant of isolation. We claim that the relation $\equiv_{L(M,\theta)}$ cannot have more than $\kappa \stackrel{\text{def}}{=} [1 + (a/\varepsilon)]^{n-1}$ classes.

This bound is established by considering a set of k words that are mutually inequivalent under $\equiv_{L(M,\theta)}$, with the aim of showing that k cannot exceed κ. This is accomplished by converting M's language-related problem to a geometric setting, by considering, for each $x \in \Sigma^\star$, the point in n-dimensional space given by $\widehat{\Delta}(\boldsymbol{q}_0, w)$ (cf. Eq. 5).

In the language-related setting, we consider an arbitrary pair of inequivalent words, $x_i, x_j \in \Sigma^\star$, and note that there must exist $y \in \Sigma^\star$ such that (w.l.o.g.) $x_i y \in L(M, \theta)$ while $x_j y \notin L(M, \theta)$. In the geometric setting, this translates into the existence of three points:

$$\begin{array}{rl} \langle \xi_1^{(i)}, \xi_2^{(i)}, \ldots, \xi_n^{(i)} \rangle & \text{corresponding to } x_i \\ \langle \xi_1^{(j)}, \xi_2^{(j)}, \ldots, \xi_n^{(j)} \rangle & \text{corresponding to } x_j \\ \langle \eta_1, \eta_2, \ldots, \eta_n \rangle & \text{corresponding to } y \end{array}$$

such that (here are the acceptance conditions):

$$\begin{aligned} \theta + \varepsilon &< \xi_1^{(i)}\eta_1 + \xi_2^{(i)}\eta_2 + \cdots + \xi_n^{(i)}\eta_n; \\ \theta - \varepsilon &\geq \xi_1^{(j)}\eta_1 + \xi_2^{(j)}\eta_2 + \cdots + \xi_n^{(j)}\eta_n. \end{aligned}$$

Elementary reasoning then allows us to infer that

$$2(\varepsilon/a) \ \leq \ |\xi_1^{(i)} - \xi_1^{(j)}| + |\xi_2^{(i)} - \xi_2^{(j)}| + \cdots + |\xi_n^{(i)} - \xi_n^{(j)}|.$$

We next consider, for each $i \in \{1, 2, \ldots, k\}$, the set Λ_i comprising all points $\langle \xi_1, \xi_2, \ldots \xi_n \rangle$ such that

$$\bullet\ \xi_l \ \geq \ \xi_l^{(i)} \text{ for all } l \in \{1, 2, \ldots, n\} \quad \bullet \sum_{l=1}^{n} (\xi_l - \xi_l^{(i)}) \ = \ (\varepsilon/a).$$

By bounding the volumes of the sets Λ_i, and arguing that no two share an internal point, one arrives at the following bounds on the cumulative volumes of the sets.

$$kc(\varepsilon/a)^{n-1} \ = \ \sum_{l=1}^{n} \text{Vol}(\Lambda_l) \ = \ c(1 + (\varepsilon/a))^{n-1}.$$

We infer directly that $k \leq [1 + (a/\varepsilon)]^{n-1}$, as was claimed.

7 Conclusions

It has been my goal to present a technical argument for the importance of the Myhill-Nerode Theorem and the concepts it uses to characterize the notion of "state." I have attempted to do so by reviewing several applications of (the concepts underlying) the Theorem, to areas as diverse as Finite Automata theory (Section 4.1), logic design (Section 4.2), space complexity (Section 5.1), the theory of data structures (Section 5.2), and artificial intelligence/machine learning (Section 6). To the extent that the role of theoretical computer science is to provide nonobvious conceptual frameworks for thinking/reasoning about and analyzing "real" computational settings and systems—and no one can dispute that this is at least one of the roles of the theory—the Myhill-Nerode Theorem is a success story for the field, one that should be in the toolbox of every theoretical computer scientist.

In closing, I want to stress that the Myhill-Nerode Theorem and its associated concepts is just one of the treasures from the 1960's that have slipped from front stage as automata-like models have slipped from favor. I would mention the product-decomposition work in [1] as another topic in the study of state-transition systems whose significance surely transcends the study of Finite Automata in which the work originated.

Acknowledgment. It is a pleasure to acknowledge my debt to Oded Goldreich for many perceptive comments, criticisms, and suggestions. I am grateful also to several others, notably Micah Adler and Ami Litman, for sharing insights and posing technical challenges.

References

1. J. Hartmanis and R.E. Stearns (1966): *Algebraic Structure Theory of Sequential Machines*. Prentice Hall, Englewood Cliffs, NJ.
2. F.C. Hennie (1966): On-line Turing machine computations. *IEEE Trans. Electronic Computers, EC-15*, 35–44.
3. J.E. Hopcroft, R. Motwani, J.D. Ullman (2001): *Introduction to Automata Theory, Languages, and Computation* (2nd ed.). Addison-Wesley, Reading, MA.
4. J.E. Hopcroft and J.D. Ullman (1979): *Introduction to Automata Theory, Languages, and Computation* (1st ed.) Addison-Wesley, Reading, MA.
5. J. Jaffe (1978): A necessary and sufficient pumping lemma for regular languages. *SIGACT News*, 48–49.
6. R.M. Karp (1967): Some bounds on the storage requirements of sequential machines and Turing machines. *J. ACM 14*, 478–489.
7. D.E. Knuth (1973): *The Art of Computer Programming: Fundamental Algorithms* (2nd ed.) Addison-Wesley, Reading, MA.
8. H.R. Lewis and C.H. Papadimitriou (1981): *Elements of the Theory of Computation*. Prentice-Hall, Englewood Cliffs, NJ.
9. P. Linz (2001): *An Introduction to Formal Languages and Automata* (3rd ed.) Jones and Bartlett Publ., Sudbury, MA.

10. E.F. Moore (1956): Gendanken experiments on sequential machines. In *Automata Studies* (C.E. Shannon and J. McCarthy, Eds.) *[Ann. Math. Studies 34]*, Princeton Univ. Press, Princeton, NJ, pp. 129–153.
11. B.M. Moret (1997): *The Theory of Computation*. Addison-Wesley, Reading, MA.
12. J. Myhill (1957): Finite automata and the representation of events. WADD TR-57-624, Wright Patterson AFB, Ohio, pp. 112–137.
13. A. Nerode (1958): Linear automaton transformations. *Proc. AMS 9*, 541–544.
14. M.O. Rabin (1963): Probabilistic automata. *Inform. Control 6*, 230–245.
15. M.O. Rabin and D. Scott (1959): Finite automata and their decision problems. *IBM J. Res. Develop. 3*, 114–125.
16. A.L. Rosenberg (1971): Data graphs and addressing schemes. *J. CSS 5*, 193–238.
17. A. Schönhage (1980): Storage modification machines. *SIAM J. Computing 9*, 490–508.
18. M. Sipser (1997): *Introduction to the Theory of Computation*. PWS Publishing Co., Boston, MA.
19. A.M. Turing (1936): On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.* (ser. 2, vol. 42) 230–265; Correction *ibid.* (vol. 43) 544–546.
20. Wikipedia: The Free Encyclopedia (2005): `http://en.wikipedia.org/wiki/Pumping_lemma`