

Approximate Recognition of Non-regular Languages by Finite Automata

Gerry Eisman†

B. Ravikumar††

Department of Computer Science†
San Francisco State University
San Francisco, CA 94132

Department of Computer Science††
Sonoma State University
Rohnert Park, CA 94128

Abstract

Approximate computation is a central concept in algorithms and computation theory. Our notion of approximation is that the algorithm perform correctly on *most* of the inputs. We propose some finite automata models to study the question of how well a finite automaton can approximately recognize a non-regular language. On the one hand, we show that there are natural problems for which a DFA can correctly solve almost all the instances. The design of these DFA's leads to a linear time randomized algorithm for approximate integer multiplication. On the other hand, we show that some languages (such as $L_{majority} = \{x \in (0+1)^* \mid x \text{ has more 1's than 0's}\}$) can't be approximated by any regular language in a strong sense. We also present results comparing different models of approximation.

Keywords: finite automata, approximation, majority language, squaring

1 Introduction

A motivating problem for this work is the following: Consider the language $L_{majority} = \{x \in (0+1)^* \mid x \text{ has more 1's than 0's}\}$. This language is not regular and hence can't be recognized by a DFA (Hopcroft 1979). We want to design a DFA that recognizes as many instances of this language correctly as possible. The number of strings of length n in $L_{majority}$ tends to 50% as $n \rightarrow \infty$ so a DFA with one state that just accepts all the strings in $(0+1)^*$ (or rejects all of them) has a success rate of 50% over strings of length n as $n \rightarrow \infty$. Suppose we are allowed to create a much more sophisticated DFA with, say, one million states. Can we improve the success rate beyond 50%? We show in Section 4 that this is not the case. No matter how large the DFA is, its success rate on the instances of the language $L_{majority}$ can never be $(50 + \epsilon)\%$ or better for any $\epsilon > 0$ (for large n).

Another motivation for this study is to identify situations in which finite automata can be used to develop efficient algorithms. There are some efficient string matching algorithms such as the KMP and the Aho-Corasick algorithm that are based on DFA (Crochemore 2002). But most of the problems are non-regular when the *yes* instances are encoded as a collection of strings. Thus, DFA model is not generally viewed as a tool for efficient algorithm design.

Copyright ©2005, Australian Computer Society, Inc. This paper appeared at The proceedings of the Twenty-Eighth Australasian Computer Science Conference (ACSC2005), Newcastle, Australia. Conferences in Research and Practice in Information Technology, Vol. 38. Vladimir Estivill-Castro, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

But this view point may change in some applications areas such as *streaming algorithms* (Muthukrishnan 2003) where rapid computation must be performed (often in a single pass), but the exact classification of strings is not a must - it is enough that the algorithm performs well most of the time. Our contention is that finite automaton based algorithm design is an appropriate paradigm in such settings. We present examples of natural non-regular languages almost of all of whose instances can be correctly processed by a DFA.

We present our result on inapproximability in the framework of Meyer and McCreight's (Meyer 1971) and Wilber's (Wilber 1983) notion of 'randomness' of a language L with respect to a complexity class C . Their definition formalizes the notion of when a language can't be approximated by any language in a complexity class C . For a language L (over alphabet Σ), let L_n be the number of strings of length n in L . Also let $A\Delta B$ denote the set of strings on which A and B disagree, i.e., $A\Delta B = (A \cap \bar{B}) \cup (\bar{A} \cap B)$. A language L is said to be *random with respect to a complexity class C* if for all $B \in C$,

$$\lim_{n \rightarrow \infty} \frac{|(L\Delta B)_n|}{|\Sigma^n|} = \frac{1}{2}$$

(Cai 1986) and (Babai 1987) applied this notion of inapproximability to the class AC^0 and showed that the parity language $L = \{x \mid x \in (0+1)^*, x \text{ has an odd number of 1's}\}$ and $L_{majority}$ can't be approximated by any language in AC^0 .

We extend the above model to quantify the notion of when a language (or a class) approximates a language. Specifically, we propose three increasingly stronger notions of approximation of one language by another. In the first model we consider the symmetric difference of the two languages as we have above:

Model I approximation: Given two languages L_1 and L_2 over the same alphabet Σ , we say that L_1 is an ϵ -*approximation* of L_2 if there exists an integer n_0 such that for all $n > n_0$, $\frac{|L_{1n}\Delta L_{2n}|}{|\Sigma^n|} \leq \epsilon$. In this case, we write $L_1 \approx_\epsilon L_2$.

Note that \approx_ϵ relation is symmetric.

Model II approximation: Given two languages L_1 and L_2 over the same alphabet Σ , we say that L_1 is a *bottom- ϵ -approximation* of L_2 if there exists an integer n_0 such that for all $n > n_0$,

1. $L_1 \subset L_2$
2. $\frac{|L_{1n}\Delta L_{2n}|}{|\Sigma^n|} \leq \epsilon$

In this case we write $L_1 < \approx_\epsilon L_2$.

Note that the $< \approx_\epsilon$ relation is not symmetric. However, we can define a similar approximation (*top- ϵ -approximation*) from above in which case we write $L_1 > \approx_\epsilon L_2$.

Model III approximation: The third model combines the $<\approx_\epsilon$ and $>\approx_\epsilon$ relations. We say that two languages L_b (bottom) and L_t (top) provide a *bounded- ϵ -approximation* of a language L if there exists an integer n_0 such that for all $n > n_0$:

1. $L_{bn} \subset L_n \subset L_{tn}$ and
2. $\frac{|L_{bn} \Delta L_{tn}|}{|\Sigma^n|} \leq \epsilon$.

In this case the "error" from below and above sum to less than or equal to ϵ . We write $L <\approx_\epsilon > < L_b, L_t >$.

In the sequel, we will be mainly interested in the case where the approximations are performed using regular languages. Thus, we will speak of the DFA acceptors of these languages as *approximators*.

We call a DFA, D an ϵ -approximator of a language L , if $L(D)$ is an \approx_ϵ -approximation of L . Similarly, D can be a *bottom- ϵ -approximator* or *top- ϵ -approximator*. Finally, D is a *bounded- ϵ -approximator* of a language L if the states of D can be classified into three types - *accepting states* A , *non-accepting states* N and *unknown states* U so that there exists an integer n_0 such that for all $n > n_0$

1. If $\delta(q_0, x) \in A$, then $x \in L$.
2. If $\delta(q_0, x) \in N$, then $x \notin L$.
3. the number of strings of length n that reach U is at most ϵ of all possible input strings of length n .

Define the density $d(L)$ of a language as:

$$d(L) = \limsup_{k \rightarrow \infty} \frac{|L_k|}{|\Sigma^k|}$$

(where Σ is the alphabet over which it is defined). Any language with density $k \geq \frac{1}{2}$ ($k \leq \frac{1}{2}$) can be k -approximated by the DFA that accepts (rejects) all the strings. For a language that is random for the class of regular languages (in the sense defined above), there is no better approximation using DFA's. On the other extreme, the strongest approximability of a language (in terms of regular languages) is the bounded ϵ -approximation with arbitrarily small $\epsilon > 0$. (We call such an approximation simply *bounded approximation*.) Our results in Sections 3 and 4 establish results of both types. Our notion of approximation is distribution dependent. i.e., our algorithm can be viewed as being correct with *high probability*, assuming that the input is uniformly distributed. The alternate notion in which randomization is built into the internal states of the automaton, but not over the input data distribution corresponds to the well-studied *probabilistic automaton* model (Paz 1971). Nonetheless, there is some connection between our notion of approximation and randomized algorithms as described below.

As a concrete application of our DFA approximators, we consider the following fundamental problem: Given two integers x and y (in decimal), compute their product (in decimal). (The choice of the base is not important.) This problem is one of the extensively studied problems (Aho 1973) for which the best known algorithm (due to Schonage and Strassen) performs $O(n \log n \log \log n)$ digit operations if the inputs are n digits long. It has been an outstanding open problem to improve this bound. We consider the approximate multiplication problem (as formally stated in Section 3) and exhibit a Las Vegas randomized algorithm (Motwani 1996) that finds an approximate integer product in expected linear time. We created a bounded approximator and converted

it into a randomized algorithm via a randomized self-reduction.

The rest of the paper is organized as follows: In Section 2, we show some general results relating the above models. In Section 3, we present a Las Vegas randomized algorithm for approximate integer multiplication based on a bounded approximator for this problem. In Section 4, we show that $L_{majority}$ is random for the class of regular languages so it can't have any good finite-state approximator. We conclude with directions for further study.

2 General Results on ϵ -Approximations

In this section we will examine the three approximation models and demonstrate that they form a hierarchy of increasing complexity. The following theorem follows immediately from our definitions.

Theorem 1.

1. If $L <\approx_\epsilon > < L_b, L_t >$ then $L_b <\approx_\epsilon L$ and $L_t >\approx_\epsilon L$
2. If $L_b <\approx_\epsilon L$ then $L_b \approx_\epsilon L$ (similarly for $L_t >\approx_\epsilon L$)
3. If $L_b <\approx_{\epsilon_1} L$ and $L_t >\approx_{\epsilon_2} L$ then $L <\approx_{\epsilon_1 + \epsilon_2} > < L_b, L_t >$

Thus, the existence of bounded- ϵ -approximations implies the existence of bottom- ϵ -approximations and top- ϵ -approximations each of which implies the existence of ϵ -approximations. Our examples below will show that this hierarchy is strict. First, a few definitions and preliminaries.

Definition. Let L be a language over Σ . Then the *density of L* is the lim sup of the proportion of strings in L of length k as k goes to infinity. That is,

$$\text{density}(L) = \limsup_{k \rightarrow \infty} \left\{ \frac{|L_k|}{|\Sigma^k|} \right\}$$

where $L_k = L \cap \Sigma^k$.

We say L is *sparse* if $\text{density}(L) = 0$. Note that for sparse languages, we may replace lim sup with a limit.

A directed graph is *strongly connected* if for each pair of nodes in the graph, q and r , there is a path from q to r and a path from r to q .

Let G be the underlying digraph of the state transition graph of a DFA. After removing nodes representing unreachable states (from the initial state) from G , the remaining connected graph may be decomposed into its strongly connected components. We now form the digraph, \hat{G} , of G , where each node of \hat{G} is a strongly connected component of G . \hat{G} is then connected and acyclic. Let B be a strongly connected component of \hat{G} . We will call B a *leaf* if there is no edge in \hat{G} from B to another strongly connected component, and we will call it *internal* if it is not a leaf.

If B is **not** a leaf, then there is at least one node in B with an outgoing arc to another component. This arc will cause B to *leak*, i.e. most paths that visit B will eventually leave never to return. We can use this observation to prove the following lemma.

Lemma 1. For each strongly connected component B of the underlying digraph of a DFA, D , let $D_{0,B}$ be the set of strings in Σ^* that determine a path from q_0 that reaches and stays in B . If B is internal, then $D_{0,B}$ is sparse.

Proof: We will use a well-known result about Markov chains (Chung 1967) to show the lemma. Recall that

in a Markov chain, a state that is not in a leaf component is called a transient state. It is a well-known fact of Markov chains that the steady state probability of any transient state is 0. This is precisely equivalent to the fact that the language is $A_{0,B}$ is sparse since the steady state probability is precisely $\frac{|A_{0,B} \cap \Sigma^n|}{|\Sigma^n|}$. This concludes the proof. Q.E.D.

The converse of this lemma also holds, i.e. if B is a leaf component then $A_{0,B}$ is not sparse. In fact more can be said as shown in the next lemma which will be used in the sequel.

Lemma 2. Let A be a minimal DFA. If b is a state in a leaf component B , then $A_{0,b}$ = the strings that determine paths from q_0 to b is not sparse.

Proof: Since G is connected (A is minimal) there is at least one path from q_0 to b . Let w_b be a word labeling one such path and suppose w_b has length c . Since B is a leaf, then all paths from b stay in B . Let d be the maximum distance from any node in B to q_i . Then for each node r in B there is at least one word of length $\leq d$ which determines a path from r to b . Choose one such word, w_{rb} , for each r . Consider the set S_t = set of all words in Σ^* whose lengths range from $c+t$ to $c+t+d$. Let x be an arbitrary string of length t , and suppose the path starting at b determined by x ends at r . Then the string $w_b x w_{rb} \in S_t$ determines a path from q_0 that reaches b travels to r and then back to b . Thus, S_t contains at least 2^t words of $A_{0,b}$. As t increases, this is a non-vanishing fraction of all words in S_t , and therefore $A_{0,b}$ is not sparse. Q.E.D.

Moreover, for sparse regular languages, we can prove a stronger result. This result will be used later in Section 4.

Lemma 3. Let L be a sparse regular language over an alphabet Σ . Then there exists a real number $\beta < |\Sigma|$ and an integer n such that for all $k \geq n$, $|L_k| < \beta^{k+1}$.

Proof: For simplicity, let $\Sigma = \{0, 1\}$. Let A be a minimal DFA accepting L . By Lemma 2, we may assume that no final states of A belong to leaf components.

Now we can apply Lemma 1. Let q_f be a final state of A belonging to an internal component B , and let \hat{B} be the set of all states from which there is a path to B , and let \hat{L} be the set of strings determining paths from q_0 that stay in \hat{B} . We will show the bound holds for \hat{L} and then the result will follow by combining the sets for each final state of A .

Since B is internal, then from each state in \hat{B} there is a path that enters B and then exits B never to return. Let d be the least integer such that for each state in \hat{B} there is one such path of length d or smaller. Then for each integer t , there is at least one string of length $t + d + 1$ which is not in \hat{L} . Thus, $|\hat{L}_{d+1+t}| \leq (2^{d+1} - 1)2^t$

Choose a real number, γ , such that

$$(1 - \frac{1}{2^{d+1}}) < \gamma^{2^{d+1}} < 1.$$

Then, for $t = 0, 1, \dots, d$, since $\gamma < 1$,

$$(1 - \frac{1}{2^{d+1}}) < \gamma^{d+1+t} < 1.$$

Now let $\beta = 2\gamma$. Then $\beta < 2$, and for $t = 0, 1, \dots, d$,

$$|\hat{L}_{d+1+t}| \leq (2^{d+1} - 1)2^t = 2^{d+1+t}(1 - \frac{1}{2^{d+1}})$$

$$< 2^{d+1+t}\gamma^{d+1+t} = \beta^{d+1+t}$$

Moreover, for $t = d + 1, \dots, 2d + 1$,

$$\begin{aligned} |\hat{L}_{d+1+t}| &\leq (2^{d+1} - 1)|\hat{L}_t| < (2^{d+1} - 1)\beta^t \\ &= 2^{d+1}(1 - \frac{1}{2^{d+1}})\beta^t < \beta^{d+1+t} \end{aligned}$$

Continuing in this manner we see that for all $s \geq d + 1$,

$$|\hat{L}_s| \leq \beta^s.$$

Finally, if we combine two sets where the first is bounded by β_1^s and the second by β_2^s with $\beta_1 < 2$ and $\beta_2 < 2$ then for a sufficiently large integer n , $\beta_1^n + \beta_2^n < 2^n$, and so if we let $\beta_3^n = \beta_1^n + \beta_2^n$, then for $k \geq 1$, $\beta_3^{nk} > \beta_1^{nk} + \beta_2^{nk}$, and so the union of the two sets is also bounded by a power function with $base < 2$. This concludes the proof.

The next theorem establishes a useful result concerning top- ϵ -approximations of the majority language.

Theorem 2. Let R be a regular language such that $L_{majority} \subset R$. Then \bar{R} is sparse.

Proof: Let D be a DFA that accepts R . In the usual way D can be partitioned into strongly connected components. As we have seen, the set of strings $L_{0,B}$ determining paths from q_0 to B is sparse if B is a non-leaf component. Let the distance between two nodes be the length of the shortest path from one to the other (distance is not symmetric) and let the diameter of a component be the longest distance between any two of its nodes. Let B be a leaf component that contains at least one final state, q_f , and one non-final state q_g , and suppose the diameter of B is k . Let u be a string of length $\leq k$ that determines a path from q_f to q_g , and let w be a string determining a path from q_0 to q_f . Then wu determines a path from q_0 to q_g , and since q_g is non-final, we have $wu \in \bar{R}$. Thus, wu has at least as many 0s as 1s, and since there at most k 0s in u , the number of 1s in w exceeds the number of 0s in w by no more than k .

We now divide the components into three categories: internal components, leaf components that contain at least one non-final state, and leaf components that consist entirely of final states. We know that the subset of $L_{majority}$ accepted by final states in the internal components is sparse. If n is the maximum diameter of all leaf components that contain at least one non-final state, then we know that the subset of $L_{majority}$ accepted at final states in these components consists of strings where the number of 1s exceeds the number of 0s by no more than n which is also a sparse set. Thus, since the density of $L_{majority}$ is $\frac{1}{2}$, most of the strings in $L_{majority}$ must be accepted by final states in leaf components which contain no non-final states. But since no arcs leave these leaf components and there are as many arcs labeled 1 as labeled 0 in any leaf, then the density of strings accepted in these leaf components that are in $L_{majority}$ will be equal to the density of strings accepted that are in $\bar{L}_{majority}$. Therefore, the density of $R \cap \bar{L}_{majority}$ is also $\frac{1}{2}$, and so we conclude that \bar{R} is sparse. Q.E.D.

Corollary 1. Let $\Sigma = \{0, 1, 2\}$ and let $L = (L_{majority}2)^*$. Then for each $\epsilon > 0$, there exists a regular bottom- ϵ -approximation of L , but for no $\epsilon < 1$ does there exist a regular top- ϵ -approximation of L .

Proof: will be presented in the final version of the paper.

The example in the corollary demonstrates that for regular approximations, the $\langle \approx_\epsilon \rangle$ relation is more strict than the $\langle \approx_\epsilon \rangle$ relation. That is, we know that $(Reg \langle \approx_\epsilon \rangle L) \Rightarrow (Reg \approx_\epsilon L)$

but that the converse is not always true. We now examine the \approx_ϵ relation. We will similarly demonstrate that

$$(Reg \approx_\epsilon L) \Rightarrow (Reg \langle \approx_\epsilon \rangle L)$$

but again, the converse does not always hold.

Let $L = \bigcup_{n \geq 0} [(L_{majority} 2)^{2n} \cup (\bar{L}_{majority} 2)^{2n+1}]$. Then using regular languages, the strings with an even number of 2s can only be approximated from below and the strings with an odd number of 2s can only be approximated from above. L can be approximated as closely as we like, but not from only one side.

In section 4 we will demonstrate that $L_{majority}$ cannot be approximated by a regular language using any of our approximation models for any $\epsilon < \frac{1}{2}$. In the next section, we turn to the question of what *can* be approximated well using a DFA model.

3 Approximate Squaring and Multiplication

First we state precisely our formulation of the approximate integer multiplication problem. Input to this problem are decimal representation of two positive integers x and y . Let ϵ , $0 < \epsilon < 1$ be a fixed real number. The output is the pair $\langle z_1, z_2 \rangle$ where z_1 is the decimal representation of the number of digits in $x \times y$, and z_2 is a decimal number that represents the leading $\lceil \log_2(\frac{1}{\epsilon}) \rceil$ bits of $x \times y$. Thus, for example, let $\epsilon = \frac{1}{8}$ and consider the input $x = 4256389$ and $y = 1414567$. Since their product is 6020947418563, the output for the problem are $z_1 = 13$ and $z_2 =$ the leading 3 digits of the product = 602. Integer squaring is the special case of this problem in which $x = y$. We present a randomized algorithm for this problem that has the following properties:

1. The output produced is always correct, i.e., the algorithm has 0-sided error.
2. The expected number of bit operations (averaged over the probability space of random bits internally generated by the algorithm) is $O(n)$ for every input of size n . i.e., the linear time complexity is guaranteed (with probability = 1) on **every** input.

We formulate approximate integer multiplication as a language recognition problem for which we will present a bounded approximator DFA's. Later we show how to design a randomized algorithm for approximate squaring and multiplication using these DFA's. The two outputs of integer multiplication problem can be defined by the languages

$M = \{[x_1, y_1] \dots [x_n, y_n] \mid x_1 x_2 \dots x_n \text{ and } y_1 y_2 \dots y_n \in (1 + 2 + \dots 9)(0 + 1 + \dots + 9)^* \text{ and } x_1 x_2 \dots x_n \times y_1 y_2 \dots y_n \text{ has } 2n - 1 \text{ digits}\}$ and

$M_j = \{[x_1, y_1] \dots [x_n, y_n] \mid x_1 x_2 \dots x_n \text{ and } y_1 y_2 \dots y_n \in (1 + 2 + \dots 9)(0 + 1 + \dots + 9)^* \text{ and } x_1 x_2 \dots x_n \times y_1 y_2 \dots y_n \text{ has } j \text{ as its most significant digit}\}$

Similar languages for integer squaring are defined in Section 3.1.

Before we present the technical details, we remark that the randomized approximate squaring algorithm presented in this section is not just of theoretical interest. There are applications in which this algorithm can replace the exact integer multiplication algorithm with significant performance improvement. One such

scenario is described in (Ravikumar 2003), (Ravikumar 2004).

Remark: It should be noted that there are two distinct notions of approximation we are using in this section. In the formulation of the problem, we introduce approximation by requiring the computations of only the leading digits. The recognizing DFA introduces further approximations by being correct only on most of the inputs, and not all of them. But it seems both approximations are crucial for establishing the result. The reason for the latter is that the languages stated above are not regular, as can be shown using the pumping lemma. The former is an interesting open problem. (See the concluding section.)

3.1 A bounded approximator DFA for Approximate Squaring

In this section, we will consider approximate squaring problem. Let j be a positive integer. Consider the languages

$L' = \{x_1 \dots x_n \mid x_1 \dots x_n \in (1 + 2 + \dots 9)(0 + 1 + \dots + 9)^* \text{ and } x_1 x_2 \dots x_n^2 \text{ has } 2n - 1 \text{ digits}\}$ and

$L_j' = \{x_1 \dots x_n \mid x_1 x_2 \dots x_n \in (1 + 2 + \dots 9)(0 + 1 + \dots + 9)^* \text{ and } x_1 x_2 \dots x_n^2 \text{ has } j \text{ as a prefix}\}$

Fix a real number $\epsilon > 0$. We will describe a bounded approximator for both L' and L_j' .

Lemma 4. There exists a bounded approximator for L' .

Proof. Let $x = x_1 \dots x_n > 0$ be an n -digit decimal integer without leading zeroes. Let the decimal representation of $\sqrt{10}$ be $y_0.y_1 y_2 \dots$. Thus, $y_1 = 3$, $y_2 = 1$, $y_3 = 6$, $y_4 = 2$ etc. It is easy to see that x^2 has $2n - 1$ or $2n$ digits according as $x_1 \dots x_n \leq y_1 y_2 \dots y_n$ or $x_1 \dots x_n > y_1 y_2 \dots y_n$. For example, if $x = 3161$, then since $3161 < 3162$ the first four digits in the decimal representation of square root of 10, we see that $3161^2 = 9991921$, which has $2n - 1 = 7$ digits. But $317 > 316$, the first three digits of square root of 10, and we see that $317^2 = 100489$, which has $2n = 6$ digits. Let $k = \lceil \log_{10}(\frac{1}{\epsilon}) \rceil$. We create a DFA M that accepts a string x if its first k digits form an integer that is less than $y_1 y_2 \dots y_k$, rejects the string x if its first k digits form an integer that is greater than $y_1 y_2 \dots y_k$ and the DFA takes the string x to the only state in U (don't know state) if the prefix of length k of x is $y_1 y_2 \dots y_k$. It is easy to see that the size of this DFA is $O(k)$. The proof that M is a ϵ -approximator for L' is obvious. This concludes the proof. Q.E.D.

Lemma 5. There exists a bounded approximator for L_j' for each j .

Proof. Let the decimal representations of the integers \sqrt{j} , $\sqrt{j+1}$, $\sqrt{10j}$ and $\sqrt{10(j+1)}$ be:

\sqrt{j} be $v_0.v_1 v_2 \dots$

$\sqrt{j+1}$ be $x_0.x_1 x_2 \dots$

$\sqrt{10j}$ be $y_0.y_1 y_2 \dots$ and

$\sqrt{10(j+1)}$ be $z_0.z_1 z_2 \dots$

Let $k = \lceil \log_{10}(\frac{1}{\epsilon}) \rceil + 2$. Let $pre_k(w)$ denote the prefix of w of length k . Let w be an n -digit integer. w^2 has j as the prefix if and only if $v_0 v_1 v_2 \dots w_{k-1} < pre_k(w) < x_0 \dots x_{k-1}$ or $y_0.y_1 y_2 \dots < pre_k(w) < z_0.z_1 z_2 \dots$. It is easy to see that a DFA M_j with $O(k)$ states can be built to check if w satisfies one of the conditions. Uncertainty arises if $pre_k(w)$ exactly matches one of the four strings $v_0 v_1 v_2 \dots v_{k-1}$, $x_0 \dots x_{k-1}$, $y_0 y_1 y_2 \dots y_{k-1}$ or $z_0 z_1 z_2 \dots z_{k-1}$. It is easy to check that the probability of uncertainty (when w

is chosen uniformly among strings of length n) is at most ϵ . Q.E.D.

Note that we can combine the above recognizers into a single DFA that would output (via the final state reached) the leading k -digits of x^2 for any input string x . But the number of states in the resulting DFA would be a polynomial in $\frac{1}{\epsilon}$, which is too large in practice. However, a more effective way to design an algorithm for the approximate squaring problem is to build a digital search tree in which each node is associated with a DFA which can be built on the fly. The details will be presented in the full version of the paper.

Now we show the main result of this subsection.

Theorem 3. There is a Las Vegas randomized approximation algorithm for integer squaring problem stated above. This algorithm performs $O(n)$ operations on an n -digit input.

Proof. Note that the model of Computation we use for this algorithm is not a DFA but a random access machine (RAM) (Aho 1973). Also, the algorithm's performance is independent of the input distribution. The basic idea is to reduce the task of computing x^2 (where x is an n -digit input) to the computation of squares of "random" integers of roughly the same size and use the DFA's described in the above lemmas to solve these instances. In the following, we use $Size$ and Msd to denote calls to DFA's that compute the length and the leading k digits of the square of the input string. More specifically, $Size(t)$ returns the number of digits in t^2 and $Msd(t, k)$ returns the leading k digits of t^2 . In the description of the following algorithm, when we make calls to $Size$ and Msd , it is possible that the output is undetermined (because the state reached by the DFA is in U .) In such cases, the implicit rule is that the computation is abandoned and restarted from the beginning. Eventually, the algorithm will find a path in which the computation terminates. The resulting algorithm always terminates in a finite number of steps, with the correct answer. The only random variable is the time complexity. We show that the expected time complexity is $O(n)$, with probability 1, on every input of length n .

First we consider the problem of computing z_1 , the number of digits in x^2 . This is part 1 of the algorithm. (Part 2 involves computing the leading k -digits of x^2 , which is described later. On input x , the algorithm randomly generates an integer y such that $|x - y|$ has n digits. (One simple way to do this is as follows: Repeatedly generate a random integer y until a y is found to meet the requirement. It is easy to see that the probability of success at each step of the iteration is at least 0.2 and hence the expected number of iterations is 5, as the number of iterations is geometrically distributed.) The algorithm proceeds in two phases. In phase 1, the number of digits in $x^2 + y^2$ is determined. In phase 2, the number of digits in x^2 is determined.

The algorithm computes $P = x + y$ and $Q = |x - y|$. (Note that P and Q can be computed in linear time.) It then computes $p = Size(P^2)$, $q = Size(Q^2)$ and $r = Size(R^2)$ using the DFA described in Lemma 1. We observe that $p \in \{2n - 1, 2n, 2n + 1\}$ and $q, r \in \{2n - 1, 2n\}$. There are two cases to consider.

Case 1: $p = q$. In this case, the number of digits in $x^2 + y^2$ is p ($= q$). (The reason is that $|x - y|^2 \leq x^2 + y^2 \leq (x + y)^2$.)

Case 2: $p \neq q$. In this case, $p \geq q$, and let $d = p - q$. The algorithm uses a constant k that is predetermined based on ϵ . The algorithm calls $Msd(P, k)$ and sets the value to a . It calls $Msd(Q, k)$ and sets the value

to b . Now we consider two subcases:

Case 2.1. If $d = 1$ then then if $10p + q$ has 1 as the leading digit then $x^2 + y^2$ has $2n$ digits; else it has $2n + 1$ digits.

Case 2.2. If $d = 2$ then if $100p + q$ has 1 as the leading digit then $x^2 + y^2$ has $2n$ digits; else it has $2n + 1$ digits.

Note that in the above cases, finding the leading digit of $10p + q$ or $100p + q$ does not require a call to $Msd(\cdot)$ since these are k digit numbers (k is a constant), not n digit numbers. So these additions can be performed in constant time.

Now we have computed the number of digits in $x^2 + y^2$; call it s . Recall that the number of digits in y^2 is r . In phase 2, the algorithm computes the number of digits in x^2 as follows: If $x^2 + y^2$ has $2n - 1$ digits, then clearly x^2 has $2n - 1$ digits. If $x^2 + y^2$ has $2n$ digits, then we consider two cases depending on the number of digits in y^2 .

Case 1. y^2 has $2n$ digits. The algorithm calls $Msd(y, 1)$ to compute the leading digit of y^2 . Call it m . The algorithm computes the leading digit m' of $x^2 + y^2$ (using the second part of the algorithm described below.) It announces $2n$ as the output if $m \neq m'$; else it announces $2n - 1$ as the output.

Case 2. y^2 has $2n - 1$ digits. The algorithm calls $Msd(y, 1)$ to compute the leading digit of y^2 . Let m be the leading of y^2 . The algorithm computes the leading two digits m_1 and m_2 of $x^2 + y^2$ (using the second part of the algorithm described below.) If $m_1 > 1$, then it announces $2n$ as the output. If $m_1 = 1$, then if $m_2 > m$ then the $2n$ is declared the output; else it announces $2n - 1$ as the output.

It is not hard to see that the algorithm is correct. In the final version of the paper, we will show that the expected number of bit operations performed by the algorithm is $O(n)$ on inputs of length n .

The second part of the algorithm is to compute the leading k -digits of x^2 . The algorithm uses the same y randomly generated in part (1). We will also use P , Q , p , q and r computed in the previous part of the algorithm. The algorithm calls $Msd(x + y, 2k + 2)$ to find A' , the leading $2k + 2$ digits of $(x + y)^2$. It calls $Msd(|x - y|, 2k + 2)$ to find B , the leading $2k + 2$ digits of $|x - y|^2$. Now we can compute the leading $k + 1$ digits of $x^2 + y^2$ as follows: Recall $p =$ the size of P^2 , and $Q =$ size of q^2 . If $p = 1 + q$, then set $A' = 10 * A$, else $A = A'$. The leading $k + 1$ digits of $A + B$ (call it C') gives the leading $k + 1$ digits of $2(x^2 + y^2)$ (except in the case in which the lowest k digits of $A + B$ is exactly equal to $10^k - 1$, an event of probability $\frac{1}{10^k - 1}$. In this case, the computation is repeated with a different random y . Finally, we get C , the leading k digits of $x^2 + y^2$ as follows: If the leading digit of C' is not 1, then $C = \lfloor C'/20 \rfloor$, else $C = \lfloor C'/2 \rfloor$.

Finally, determine $D =$ the leading k digits of y^2 using a call to $Msd(y, k)$. The leading k digits of x^2 is $D - C$. The proof of correctness of the algorithm, as well as the proof that the expected number of steps performed by the algorithm is $O(n)$ are not difficult, but tedious. The former follows from the identity $x^2 = \frac{1}{2}[(x + y)^2 + |x - y|^2] - y^2$. The latter follows from the fact that when y is chosen at random (subject to a mild restriction about the length $|x - y|$, $(x + y)^2$, $|x - y|^2$ and y are "random" (although they are not uniformly distributed over all possible n digit integers). Thus, the performance of the algorithm is independent of x . Further, it can shown that with a constant probability $c > 0$ (independent of n), in a

single iteration, both parts of the algorithm will complete without uncertainty. Thus, in expected $O(n)$ steps, the algorithm terminates with probability 1. Q.E.D.

3.2 Approximate Multiplication

In this section, we will use the results of the previous subsection to show that a ϵ -approximator exists for integer multiplication. Recall the definitions of M' and M_j defined in the beginning of this section.

If the integers $x_1\dots x_n$ and $y_1\dots y_n$ do not have the same length, we assume that some trailing special symbol $-$ has been added to make the two numbers to have the same length. Thus, for example, if $x = 234154$ and $y = 15670$, then the encoding of the pair would be $[2, 1][3, 5][4, 6][1, 7][5, 0], [4, -]$. The reason for padding the shorter string with trailing $-$'s rather than with leading 0's will become clear in the discussion below.

First, we show a lemma that reduces the problem of finding the number of digits in a product xy to that of determining the leading digit of xy .

Lemma 6. Let $x \geq y$, and let the number of digits in x be n , and the number of digits in y be m . The number of digits in xy is either $n + m$ or $n + m - 1$. Further, the number of digits in xy is $n + m$ if and only if the leading digit of xy is less than the leading digit of x .

Proof. will be presented in the full version of the paper.

Theorem 3. There exists a bounded approximating DFA for the languages M and M_j .

Proof. (outline) By the above lemma, it is clear that a bounded approximator for M_j (where j is a single digit integer) can be easily converted to a bounded approximator for M . In the rest of the proof, we will consider M_j . For simplicity, we will focus on single digit j . The basic idea is to use the DFA approximator for L_j and L' to determine the number of digits in $(x + y)^2$, x^2 and y^2 , as well as the leading digits in these numbers. Using the identity, $xy = \frac{1}{2}[(x + y)^2 - (x^2 + y^2)]$, we determine the leading digit of xy . There is one problem that needs to be considered. If x and y have significantly varying lengths, the leading several digits of $(x + y)^2$ and $x^2 + y^2$ will be identical and hence we can't determine the leading digit of $2xy$. To avoid this problem, we have chosen an encoding in which the string is padded with trailing special symbol $-$ so that the two strings have the same length. If we treat the $-$ as 0's, in effect, we have multiplied by the smaller number by 10^k where k is the difference between the number of digits in the two strings. Clearly, the leading digits of the product will not be affected by this scaling. The details are similar to the algorithm described in the previous section in which the identity $x^2 = \frac{1}{2}[(x + y)^2 + |x - y|^2] - y^2$ was used to determine the leading digits of x^2 . The only difference between the two is the minor difference in the identity, and the fact that the approximator in this theorem is a DFA. But a DFA solution is possible since the current theorem, the algorithm is not required to be a Las Vegas randomized algorithm. Q.E.D.

We now state the main theorem of this section.

Theorem 4. There is a Las Vegas randomized approximation algorithm for integer squaring problem stated above. This algorithm performs $O(n)$ operations on an n -digit input.

The proof is based on the techniques presented above. But the details are somewhat complicated and tedious so we defer them to the full version of the paper.

4 Majority is Random for the class of Regular Languages

Throughout this section will assume that $\Sigma = \{0, 1\}$. We now examine the problem of approximating the language $L_{majority}$ by regular languages. We begin with the following definitions.

Definition: A string, $w \in \Sigma^*$ is 1-dominant if it has more 1s than 0s. Similarly a string may be 0-dominant or 01-equal. Given a finite set of strings, F , let

$n_1(F)$ = number of strings in F which are 1-dominant. $n_0(F)$ is similarly defined.

For infinite languages we may consider the lim sup of the proportion of strings equal to a given length that are 1-dominant.

Definition: For any language, L , we define the 1-density of L (if the limit exists) by

$$d_1(L) = \limsup_{k \rightarrow \infty} \frac{n_1(L_k)}{|L_k|}$$

where L_k is the set of strings in L of length k . Again, $d_0(L)$ is similarly defined.

As we shall see, approximating 1-density depends on both the number of 1-dominant strings in L and on the density of L itself relative to Σ^* .

We are particularly interested in regular languages. For example, consider the language $L = (1110)^* + (1101)^* + (1000)^*$. Clearly the 1-density of L is $\frac{2}{3}$. But notice that L is sparse. It contains no string whose length modulo 4 is congruent to 1, 2 or 3, and even considering strings whose length is a multiple of 4, as length increases, L contains a decreasingly small fraction of all possible strings. Or consider the example, $M = 1(11 + 10 + 01 + 00)^*$. M contains only odd length strings, in fact, all odd length strings that begin with 1. Thus, $density(M) = \frac{1}{2}$. If we examine the distribution of strings in M_k determined by the number of 1s in each string, we obtain a binomial distribution, and, as k increases, the impact of the lead 1 on this distribution will be negligible, and so we see that $d_1(M) = \frac{1}{2}$.

It should also be noted that if $d(L) = 1$, then for some infinite sequence $\{k_i\}$, L_{k_i} contains most of the strings of Σ^{k_i} , and since about half of these strings are 1-dominant. we have $d_1(L) = \frac{1}{2}$. Similarly, if L is sparse, then $density(\bar{L}) = 1$ and so $d_1(\bar{L}) = \frac{1}{2}$. Next we combine the proportion of 1-dominant strings in L with the 0-dominant strings in \bar{L} . That is, let

$$D(L) = \lim \sup_{k \rightarrow \infty} \frac{(n_1(L_k) + n_0(\bar{L}_k))}{2^k}$$

The main theorem of this section is:

Theorem 5. $L_{majority}$ does not have a regular \approx_ϵ -approximation for any $\epsilon < \frac{1}{2}$. (Equivalently, $L_{majority}$ is random for the class of regular languages.)

The rest of the section is devoted to proving this theorem. If we consider a DFA A , which accepts the language L , then we can look at $D(L)$ as a measure of how well A recognizes 1-dominant strings. The limit expression defining $D(L)$ measures the proportion of 1-dominant strings of length k that A accepts plus the proportion of 0-dominant strings that A does not

accept. In other words, if A is looking for 1-dominant strings only, then the claim is that A is correct only about $\frac{1}{2}$ of the time. Since this is what one would obtain by random guess, we can conclude that DFAs are not all that useful for recognizing the strings in the majority language. The reader may recall the earlier result of (Cai 1986) and (Babai 1987) that majority is random for the class AC^0 . Our result is incomparable to that of Cai and Babai since the class AC^0 is incomparable to the class of regular languages.

The proof of the theorem relies on properties of DFA's. The central idea is that if L is accepted by a DFA, A , then \bar{L} is accepted by the same DFA with the final states switched to non-final and vice versa. Thus, L and \bar{L} have very similar structures, and from this it can be shown that $D(L)$ and $D(\bar{L})$ are identical. Since $D(L) + D(\bar{L}) = 1$, the result follows.

In actuality, we will not be working with L and \bar{L} per se but with concatenations PM and $P\bar{M}$ where P is a sparse set, and M is a set with non-zero density. But the idea is the same. $PM \cup P(\bar{M}) = PS^*$, and we will show that $D(PS^*) = \frac{1}{2} = D(PM) = D(P\bar{M})$.

We begin by examining some properties of sparse languages.

Theorem 6. If L_1 and L_2 are two sparse regular languages, then $L_1 \cup L_2$ and L_1L_2 are sparse as well.

Proof: For the union the statement is clear. Consider DFAs, A_1 and A_2 , that accept L_1 and L_2 respectfully. Since L_1 has zero density then, by Lemma (?), the final states of A_1 must belong to an internal component of A_1 . Similarly for A_2 . From A_1 and A_2 , we can construct an automaton to accept L_1L_2 in the usual way by adding edges from final states of A_1 to states of A_2 .

This automaton is non-deterministic, but we can construct a deterministic equivalent in the usual way by considering subsets of the union of the states of A_1 and A_2 as its states. The start state of this combination is the singleton containing the start state of A_1 , and the subsets containing final states of A_2 are the final states. The sets along a path determined by an input string w , will continue to be singletons of states of A_1 unless the path passes through a final state of A_1 . At this point the set will become a pair of states, an element in A_1 and one from A_2 . The sets will remain pairs unless the path again passes through a final state of A_1 at which point the sets may become triples with one element from A_1 and at most two from A_2 . At no point will there ever be more than one state from A_1 in the set. Moreover, since L_1 is sparse, as string length increases, for most strings the A_1 element in the set will belong to a strongly connected leaf component that contains no final states of A_1 at which point the number of elements in the set will stay fixed as there are no more bifurcations.

Similarly, since L_2 is sparse, if we consider the states where the lone member from A_1 is in a leaf component (and non-final) and examine the subset of elements from A_2 , as string length increases, this set will not increase in size, and the probability that each of these elements in turn will eventually become an element of a leaf component is 1. Thus, most strings will lie outside L_1L_2 , and so L_1L_2 is sparse. Q.E.D.

We will also need the following result on the rate at which the proportion of strings of a given length in a sparse regular language diminishes with length.

Lemma 7. Let L be a regular language with non-zero-density and let Q be a non-empty regular sparse language such that for each w in LQ , w has a unique factorization into $w = uv$ with u in L and v in Q . Then $d_1(LQ) = d_1(L)$. The same result holds for the

product QL , as long as unique factorization applies.

Proof: Let x be a string of length n chosen uniformly from Σ^n . The number of 1's in x follows a binomial distribution. We refer to this distribution as the one distribution. For each positive integer r , as n increases the proportion of strings that fall in the middle of the distribution, between $n/2 - r$ and $n/2 + r$ will tend to 0. We refer to this middle as $Mid_{n,r}$. Similarly, since L has non-zero density, then in L_n , this middle will tend to 0. n and r will be chosen deliberately below.

The idea of the proof is to divide $(LQ)_n$ into two pieces. By the assumption of unique factorization we form the disjoint union $(LQ)_n = \bigcup_{k=0, \dots, n} L_k Q_{n-k} = U_1 \cup U_2$ where

$$U_1 = \bigcup_{k=0, \dots, n-r} L_k Q_{n-k} \quad \text{and} \quad U_2 = \bigcup_{k=n-r+1, \dots, n} L_k Q_{n-k}$$

In U_1 we will use the bound from Lemma 6 to show that it is sparse, and in U_2 we use the fact that the middle of the one distribution vanishes to show that since Q is sparse, the strings of Q have little impact on the one distribution of strings in LQ . This distribution will be almost identical to the one distribution of strings in L .

First we need to establish that LQ has non-zero density. Suppose $density(L) = p > 0$. Then for every δ , there exists an integer K , and an infinite sequence $\{k_i\}$ with $k_i > K$, $\frac{|L_{k_i}|}{2^{k_i}} > (p - \delta)$. Suppose $w \in Q$ (Q is non-empty) and $|w| = r$. Let $(LQ)_n$ be the set of strings of length n in LQ . Then if $n > K + r$, and $n \in \{k_i\}$, $(LQ)_n$ contains at least $(p - \delta)2^{n-r}$ strings and so $\frac{|(LQ)_n|}{2^n} > \frac{(p-\delta)}{2^r}$, and so $density(LQ)$ is at least $\frac{p}{2^r}$. Thus, LQ is not sparse.

We examine U_1 first. Since Q is sparse, then by Lemma 6, there exists a real number $1 \leq \beta < 2$ and an integer K_1 such that if $k > K_1$, then $|Q_k| < \beta^k$. We first choose r to be greater than K_1 . Then, since $|L_k| \leq 2^k$, we have

$$|U_1| < \beta^n + 2\beta^{n-1} + \dots + 2^{n-r}\beta^r$$

$$= \frac{\beta^n((\frac{2}{\beta})^{n-r+1} - 1)}{((\frac{2}{\beta}) - 1)}$$

$$\text{Let } \beta_0 = \frac{1}{((\frac{2}{\beta}) - 1)} \text{ and so } |U_1| < \beta_0 \beta^{r-1} 2^{n-r+1}.$$

Then $\frac{|U_1|}{2^n} < 2\beta_0(\frac{\beta}{2})^r - 1$. By increasing r (we will increase n along with it) we can make this bound as small as we like. Thus, U_1 has zero density, and so even if all the strings in U_1 are 1-dominant, then since LQ has non-zero density, the contribution that they make to $d_1(LQ)$ is negligible. We may choose ϵ to be arbitrarily small and determine r so that this contribution is less than $\frac{\epsilon}{2}$.

We now fix r and turn our attention to U_2 . Suppose L has non-zero density p . As string size increases, then for an infinite subsequence, $\{k_i\}$, the ratio, $\frac{|L_{k_i}|}{2^{k_i}}$ approaches p . We can choose a lower bound on k_i to approximate this ratio as closely as we like. $Mid_{n,r}$ can be made as small as desired by increasing n . Holding r fixed, we increase n until the proportion in this middle range is less than $\frac{p\epsilon}{2^r}$. Let N_1 be the minimum such n , and let $N = N_1 + r$. If $n > N$, then $n - r > N_1$, and if $k > n - r$, then the middle part of the distribution of strings in L_k is less than $\frac{p\epsilon}{2^r}$ of all strings of length k .

There are r sets that form the union U_2 . In each set, $L_k Q_{n-k}$, the strings in Q_{n-k} have length less than r . Thus, even if a string in Q_{n-k} consists entirely of 0's or 1's, it will not change the dominance of 0's or 1's in a string w in L_k unless the absolute value of the difference between the number of 0's and 1's in w is less than r . Therefore, the strings in Q_{n-k} will

at most impact the middle part of the distribution of the strings in L_k . Thus, $n_1(L_k Q_{n-k}) \leq (n_1(L_k) + Mid_{k,r})|Q_{n-k}|$.

By the assumption of unique factorization we know that $|L_k Q_{n-k}| = |L_k||Q_{n-k}|$, and so

$$\frac{n_1(L_k Q_{n-k})}{|L_k Q_{n-k}|} \leq \frac{(n_1(L_k) + Mid_{k,r})}{|L_k|}$$

$$= \frac{n_1(L_k)}{|L_k|} + \frac{Mid_{k,r}}{|L_k|}$$

Since $\frac{n_1(L_k)}{|L_k|}$ approaches $d_1(L_k)$, $|L_k|$ approaches $p2^k$, and we chose k large enough so that $\frac{|Mid_{k,r}|}{2^k} < \frac{\epsilon}{2^r}$, then if we sum the r sets in U_2 we see that U_2 , like U_1 , contributes less than $\frac{\epsilon}{2}$ to $d_1(L_Q)$ as well, and this proves the lemma.

It may be noted that this result can be extended to products PLQ where both P and Q are sparse and regular and L is regular but not sparse, as long as unique factorization holds.

Let A_{0B} strings that determine paths in the accepting DFA A , that lead from q_0 to a state in B , where B is a leaf component. Let b_1, b_2, \dots, b_t be the entry points of B , i.e. the nodes of B for which there is an incoming edge from some other strongly connected components, and let b be any node of B . Let A_{0b} be the set of strings determining paths from q_0 to b . A_{0b} may be divided into the disjoint union $i = 1, \dots, t$ of $P_{0b_i} A_{b_i b_i} Q_{b_i b}$ where P_{0b_i} is the set of prefixes that lead from q_0 to b_i as the first node reached in B , $A_{b_i b_i}$ is the set of strings that determine paths from b_i to itself, and $Q_{b_i b}$ is the set of suffixes determining paths from b_i to b that do not pass through b_i again. (If b itself is b_i , then $Q_{b_i b}$ consists only of the empty string. Similarly if b_i is q_0 , then P_{0b_i} consists entirely of the empty string.) Note that since A is deterministic, each string in $P_{0b_i} A_{b_i b_i} Q_{b_i b}$ can be uniquely factored into a string in P_{0b_i} followed by a string in $A_{b_i b_i}$ followed by a string in $Q_{b_i b}$. Unique factorization plays a role in the results below. We also note that P_{0b_i} is sparse since if $b_i \neq q_0$ then all arcs coming into b_i come from nodes in internal components, and thus P_{0b_i} is the union of sparse languages.

We now consider a second node c of B . A_{0c} has the same structure as A_{0b} . The only difference is that we exchange the set of suffixes, $Q_{b_i c}$ for $Q_{b_i b}$.

Since B is a leaf, we have $A_{0B} = \cup_i P_{0b_i} \Sigma^*$ and $\Sigma^* = \cup_{b \in B} A_{b_i b_i} Q_{b_i b}$. A DFA H that accepts $Q_{b_i b}$ can be constructed from B by dividing state b_i into two states, b_{iout} and b_{iin} where b_{iout} is incident to all arcs in B that come out of b_i , and b_{iin} is adjacent to all arcs in B that come into b_i . b_{iout} is the initial state of H , and b is the only final state. Since B is a strongly connected component, there is a path in H from b to b_{iin} , but since b_{iin} has no arcs coming out, it belongs to a separate strongly connected component of H than b , and so the component of H containing b is not a leaf. Therefore, from Lemma 1, it follows that $Q_{b_i b}$ must be sparse. Since $\Sigma^* = \cup_b A_{b_i b_i} Q_{b_i b} = (A_{b_i b_i}) \cup_b Q_{b_i b}$ is not sparse, and since the union of sparse languages is sparse, it follows that $A_{b_i b_i}$ must have non-zero density.

Finally, we are ready to prove

Theorem 7. For all regular languages L , $D(L) = \frac{1}{2}$

Proof: Let B be a leaf component graph of the DFA that recognizes L . We know from Lemma 7 that since P_{0b_i} and $Q_{b_i b}$ are sparse, then $d_1(P_{0b_i} A_{b_i b_i} Q_{b_i b}) = d_1(A_{b_i b_i})$. Moreover, since Σ^* is the disjoint union $\cup_{b \in B} A_{b_i b_i} Q_{b_i b}$ and $d_1(\Sigma^*) = \frac{1}{2}$, then $d_1(A_{b_i b_i}) = \frac{1}{2}$ for all b_i . Finally, if we create the union of sets of the form, $P_{0b_i} A_{b_i b_i} Q_{b_i b}$ where b is a final state of the automaton, and a member of a leaf component (we can safely ignore the internal components since

those sets have zero density), then we have our result. Q.E.D.

Theorem 5, the main theorem of this section, is a direct consequence of Theorem 7.

5 Conclusions and Direction for Future Work

We hope that the models presented in this paper would lead to further results and a better understanding of when a non-regular language be approximated by a regular language. It would be interesting to identify structural properties that make a language random for the class of regular languages. It is also of interest to study approximation by other models such as push-down automaton or counter machines. We suspect that for any $\epsilon > 0$, there are context-free languages that have a bounded ϵ -approximator for $\epsilon \geq \delta$, but not for $\epsilon < \delta$. We are also interested in decidability results such as whether a given context-free language admits a \approx_ϵ approximator for all $\epsilon > 0$. Of course, we are also interested in using the approximator models proposed in this paper to design approximation algorithms for other related problem. For example, we may consider the problem of determining the middle digit (instead of the leading digit) of x^2 for a given input x . (To be concrete, when there are two middle digits, the output should be the right one.) It appears that a linear time randomized algorithm for this problem, if at all possible, would require substantially different techniques.

References

- A.Aho, J.Hopcroft and J.Ullman, *Design and Analysis of Computer Algorithms*, Addison-Wesley, 1973.
- L.Babai, A random oracle separates PSPACE from the polynomial hierarchy. *Information Processing Letters*, 26:51-53, 1987.
- Jin-yi Cai, With Probability One, A Random Oracle Separates PSPACE from the Polynomial-Time Hierarchy *ACM Symposium on Theory of Computing*, 21-29, 1986.
- K.L.Chung, *Finite Markov Chains*, Springer-Verlag, Inc. 1967.
- M.Crochemore and Rytter, *Jewels of Stringology*, World-Scientific, Inc. 2002.
- J.Hopcroft and J.Ullman, *Introduction to Automata, Formal Languages and Theory Computation*, Addison-Wesley, Inc. Reading, MA, 1979.
- R.Motwani and P.Raghavan, *Randomized Algorithms*, Cambridge University Press, 1996.
- A.R.Meyer and McCreight, Computationally complex and pseudo-random zero-one valued functions. *Theory of Machines and Computations*, Editors: Z.Kohavi and A.Paz, 19-41, Academic Press, 1971.
- S. Muthukrishnan, *Data Streams: Algorithms and Applications* (unpublished notes), 2003.
- A.Paz, *Probabilistic automata*, Academic Press, 1971.
- B. Ravikumar, *What are the leading 10 digits of $666!^{666!}$? A problem in Mathematics of Oz by Clifford Pickover* unpublished manuscript that can be accessed from <http://linux.cs.sonoma.edu/~ravi/cliff666fact.html>, 2003.

- B. Ravikumar, *A Las Vegas Randomized Algorithm for Approximate Exponentiation* (submitted for publication), 2004.
- R. E. Wilber Randomness and the Density of Hard Problems *IEEE Conf. on Foundations of Computer Science*, 335-342, 1983.