Sprees, a Finite-State Orthographic Learning System that Recognizes and Generates Phonologically Similar Spellings

by

Latanya Sweeney

ALB Computer Science
Harvard University

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
AT THE
MASSACHUSETTTS INSTITUTE OF TECHNOLOGY

MAY 1997

Signature of Author:_____
Department of Electrical Engineering and Computer Science
May 15, 1997

Certified by:_____
Peter Szolovits
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by:_____
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Sprees, a Finite-State Orthographic Learning System that Recognizes and Generates Phonologically Similar Spellings

by

Latanya Sweeney

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 1997 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Electrical Engineering and Computer Science

ABSTRACT

We present a computer learning program named Sprees that "learns" by building a deterministic finite state automaton that represents orthographic generalizations of the spellings it encounters. It then uses the automaton to recognize and generate phonologically similar spellings beyond those in the initial training set. The saying, "i before e except after c," demonstrates the kind of spelling regularity Sprees would learn after being exposed to words like "receive" and "thief."

Given a list of common female first names, for example, Sprees can generate additional names like "Erisa" and "Kathel" that were not included in the training words but were identified by all the native English speakers we tested as sounding like female names. In commercial advertising, Sprees can help produce new words to use as company and product names; these words would suggest meaning while still matching spelling to common pronunciation. As a recognizer, Sprees has already proven itself invaluable in maintaining patient confidentiality in medical records (Sweeney, 1996). Sprees helped to locate personally-identifying information in unrestricted text by detecting likely spellings without requiring all recognizable words be listed beforehand.

Thesis Supervisor: Peter Szolovits
Title: Professor of Computer Science and Engineering

In loving memory of

Carrie Sweeney

and dedicated to

Sylvia Barrett

Acknowledgments

I thank Professor Peter Szolovits for providing an environment that made it possible for me to learn about and explore the world of computing through my own eyes, with little financial concern and with no lack of quality time. So many times in life I found myself wanting to have the time to follow this project or that idea through to fruition but the constant demands of school, work and life thwarted my efforts. Working in Professor Szolovits' group at the Laboratory for Computer Science, in many ways, was a dream come true, though admittedly the shackles of graduate school life remained. I also thank Professor Szolovits for discussions and mentoring that extended far beyond this academic text.

Let me also take this opportunity for a more personal note. It is easy to see and then appreciate polished gold, but we are blessed to have among us those who can see value in the roughest of ordinary looking rocks. They behold beauty where we see nothing of interest, and by their faith alone they transform stone into gold for us all to see. I gratefully acknowledge ChangSook Barrett for motivating and inspiring this work. Her commitment and belief in education continue to inspire and influence all who know her, and her unwavering belief in me will never be forgotten. I also thank Henry Leitner and Harvard University DCE for their continued support and commitment to providing educational opportunities to all. Finally, but not least, I humbly thank Carrie Sweeney, SylviaBarrett, Joyce Johnson and Joseph Barrett for giving me love for a lifetime. This work has been supported by a Medical Informatics Training Grant (1 T15 LM07092) from the National Library of Medicine.

Contents

Figures

1 Introduction

Traditionally we use a list of stored words when determining whether a sequence of letters constitute an acceptable spelling. Spell correctors pose a common example where the word in question is compared against a stored dictionary of spellings to determine membership. In this view, we treat English spellings as random sequences of characters where successful membership of a test string can only be determined by listing all the members (or non-members). However, this approach denies the phonological basis of our words -- that is, written spelling stems from the spoken word. Understanding their phonological roots and orthographic patterns provides more powerful ways to recognize possible words and to generate new spellings.

We have built a computer program named Sprees that learns how spellings are formed, and then uses its acquired knowledge to identify and produce words. Sprees uses no dictionary. Instead it builds a finite-state automaton to store the spelling patterns it observes. The old adage, "i before e except after c," demonstrates the kind of spelling knowledge Sprees would acquire if trained on words such as "receive" and "thief." After sampling a small subset of English words, the program builds an automaton that in turn allows it to recognize and generate thousands of words not included in the learning sample.

Consider the problem of sharing medical records while maintaining a commitment to patient confidentiality. In medical records, letters between physicians and notes written by clinicians often contain nicknames and unusual words. Sprees was used in conjunction with stored lists to determine whether words "sounded" like medical terms, family names, first names, and so forth,

and the overall system performed extremely well in locating all references to personally-identifying information (Sweeney, 1996).

Sprees also extrapolates nonsense words, which are words that look and sound like English words but in reality have no meaning. For example, "throck" which comes from exposure to words like "throw" and "rock" is a nonsense word. In spell checking the inability of Sprees to distinguish nonsense words from meaningful words is an insurmountable weakness. Nevertheless the recognition of plausible words and the generation of new words has many uses.

In the patient confidentiality application described earlier, Sprees not only helped to locate identifying terms, but also generated fictitious names that sounded like reasonable names but were not included in any lists of known, common or uncommon names. By replacing real names with these made-up alternatives the readability of the document remained intact while the patient's real identity remained private.

In advertising and marketing the most popular method of naming products and companies is to create new words to use as names since these spellings can be registered and protected as trademarks (Nilsen, 1994). Sprees can recommend new product and company names after training on a list of words that suggest the desired connotations. The resulting words would be easy to say and remember since their spelling would match common pronunciation. Given that the United States Trademark Office has more than 800,000 active registrations with another 100,000 applications. Sprees could be quite useful in this area.

2 Background

There has been substantial research related to spell correcting (Durham, et. al., 1983) and to inference on handling errors in spelling (Carbonell and Hayes, 1983). The aims of those works however are not the same as the goals of Sprees. A spell checker should reliably identify misspellings, but one objective of Sprees is to recognize spellings that sound like words whether they are actually known English words or not. Sprees' second objective is to generate new spellings and this too lies outside the focus of work done on spell checkers.

The first spelling-to-sound data were obtained by Venezky (1963) through computer-aided analysis of the spelling-to-sound correspondences in approximately 20,000 different English words. The program produced a complete tabulation of spelling-to-sound correspondences for this corpus along with the word lists for each correspondence. This data was used in research studies to help with the teaching of spelling and reading. Venezky's system mapped graphemes, or letters and combinations of letters, to each phoneme (the basic sounds of a language). Though Sprees generates and recognizes words that "sound" similar to its list of training words, Sprees does not break words into phonemes or syllables per se. Instead Sprees uses orthographic patterns found in its training words to group letters into chunks that maintain pronunciation relationships and the chunks often fall across syllable and phoneme boundaries.

2.1 Grapheme-to-Phoneme Conversion

As part of an electronic system to convert unrestricted text to speech, Hunnicutt (1976) used a letter-to-sound approach. Decomposing words into their constituent morphemes was not sufficient for unrestricted text. It was necessary to develop a scheme for dealing with unrecognizable words. When a word could not be decomposed into its constituent morphemes, or when it appeared too infrequently in the English language to be included in the morph lexicon, the letter-to-sound system was invoked. These transcription rules were hand-crafted using extensive statistical and manual analysis of English words. In contrast, Sprees automatically deduces its own spelling patterns in a one-pass, instant learning system.

Of course Hunnicutt's goal was to produce a phonetic stream from the spelling so a computer could then speak the word. Sprees' goal in contrast, is to recognize and produce words that are believed to be pronounceable given the orthographic knowledge it extracts. In alphabetic writing systems, systematic relationships exists between letters or groups of letters in particular positions within a word and their pronunciation. Consider the words "book" and "womb." Word-initial b_ is always pronounced /b/ and word-final _mb is always pronounced /m/.

Relying on pronunciation regularities alone, we could produce a pronunciation for any letter string by assigning standard pronunciations to its orthographic constituents and concatenating. By definition, a pronunciation assembled this way will be correct for a word with regular spelling-sound correspondence, like "mint;" reasonable for a nonsense work like "rint;" and incorrect for a pronunciation exception word like "pint." See Table 1 for a summary. Pronunciation exception

words pose virtually no problem for Sprees since their spelling is regular and the human reader has the task of mapping sound utterances to the spelling. Hunnicutt's system on the other hand, was responsible for generating the proper phonemic translation in all these settings and so the resulting knowledge base was far more complex. Later, we will talk more about the relationship between phonemes and Sprees' orthographic knowledge, but first we will discuss the use of finite-state devices in language processing.

Spelling	Pronunciation	Description
mint	/m ɪ n t/	regular spelling-to-sound
rint	/r ɪ n t/	nonsense word, but regular spelling-to-sound
pint	/p aɪ n t/	pronunciation exception

Table 1. Spelling and Pronunciation.
Sprees can easily generate and recognize these three sample spellings, but a letter-to-sound system must have knowledge of pronunciation changes as is the case with "mint" and "pint." [1]

2.2 Finite-State Devices

Finite-state automata, like that used in Sprees, as well as finite-state transducers are not new. The promotion of context-free or recursively enumerable grammars as the major formalisms in language modeling is due in part to the recursive linguistic phenomena that are not finite-state (Chomsky, 1959). However, Gross (1989) reminds us that many of the known linguistic phenomena in syntax and in phonology are finite-state. So it is not surprising to find many new uses of finite-state devices emerging.

[1] All pronunciations in this paper are from The American Heritage Dictionary using the International Phonetic Alphabet.

Roche and Schabes (1995) used a finite-state transducer to build a part-of-speech tagger that operates in optimal time – out performing even the rule-based tagger that inspired it. Bird and Ellison (1994) presented a finite-state model of phonology in which automata are the descriptions and strings are the objects being described. And Roche and Schabes (1997) used a finite-state transducer to tackle a grapheme-to-phoneme conversion similar to Hunnicutt's letter-to-sound system except their version used French words. In the next sections we will describe the details of how Sprees builds and uses a finite-state automaton and then provide some run-time results.

3 Methods

Consider the following overview. During a learning phase, Sprees constructs a finite state automaton that records orthographic adjacency relationships found in the training words. Once the automaton is built, it can be used to recognize and generate spellings that were not included in the initial training set. Recognizing spellings simply involves using the automaton as an acceptor. Generating new spellings is basically a random walk on the transition graph associated with the automaton. In the next subsections we will explain the basic idea behind Sprees and then provide details for implementing Sprees as an automaton.

3.1 Basic Idea Behind Sprees

Sprees works with groups of letters called clusters. A consonant cluster is a series of one or more consonant letters, and similarly, vowel clusters consist of only vowels (a, e, i, o, u and sometimes y). An English word must contain at least one consonant cluster and one or more vowel clusters. Within a word, vowel and consonant clusters alternate. Either may start or end a spelling. The words "a" and "I" are an exception since they consist of vowels alone. No word is made from only consonants. In Sprees the letter "y" is considered a consonant, but if a spelling does not appear to have a vowel cluster and it contains the letter "y" then "y" is considered a vowel. The words "by" and "myth" are examples. Table 2 divides some sample words into their corresponding clusters.

shock	sh + o + ck
neutral	n + eu + tr + a + l
switch	sw + i + tch

Table 2. Cluster Decomposition.
Sample words divided into their corresponding vowel and consonant clusters.

Based on cluster decomposition like those in Table 2, Sprees generates a set of linked sequences that describe the relationships between adjacent clusters. These bigram-like sequences are specific to whether a cluster appears at the beginning, in the middle or at the end of the word. The sequences generated by observing the word "neutral" are shown in Table 3 as an example.

n at the beginning of a word is followed by **eu**.
eu in the middle of a word is preceded by **n** and followed by **tr**.
tr in the middle of a word is preceded by **eu** and followed by **a**.
a in the middle of a word is preceded **tr** and followed by **l**.
l at the end of a word, is preceded by **a**.

Table 3. Sprees decomposition of "neutral".
Spelling sequences deduced by Sprees when the word "neutral" was presented as a training word.

From the words: "shock," "neutral," and "switch," the program's learning was based on 3 words and the program can only recognize and generate 3 words. That is not so useful. The system gains power when the clusters overlap since the program can then generalize beyond the words in the learning sample. For example, if Sprees is given the words "race" and "late" on which to train, it will record several bigram-like sequences including the following four.

1. **a** in the middle of a word is preceded by **r**
2. **a** in the middle of a word is preceded by **l**
3. **a** in the middle of a word is followed by **c**
4. **a** in the middle of a word is followed by **t**

Learning from only those 2 words, the program can recognize and generate the 4 words: race, late, lace and rate. We continue the training of Sprees by introducing the word "may." Now, the program will recognize and generate 9 words. Likewise, adding the word "lake" to the training set allows the system to recognize and generate 12 words. The system's knowledge continues to grow in this fashion as new words are introduced. As a final example, if we add the word "map"

then the program was given only 5 words but can recognize and generate 15 spellings. See Table 4 for a summary.

race	late	may	lake	map
	lace	mace	rake	rap
	rate	mate	make	lap
		ray		
		lay		

Table 4. Learn 5 words, generate 15.
Given the 5 training words on the top row, the system can then recognize and generate 15 words. Each column shows the additional words possible when its training word is included.

We have now presented an idea of how Sprees works. It breaks words into sequences of vowel and consonant clusters that record adjacency information. These linked sequences are general enough that Sprees can use them to recognize and generate words not previously seen. In the next subsection, we will explain how Sprees is implemented.

3.2 Implementation

In computational theoretic terms, Sprees can be constructed as a deterministic finite state automaton that represents orthographic generalizations of spellings. Definition 1 contains a description of a Sprees automaton. [2] Before we look at how a Sprees automaton recognizes a spelling, it is important to realize first that the input to a Sprees automaton is not the actual spelling. Instead the original text string is pre-processed to decompose the spelling into an

[2] See Sipser(1996) for a thorough coverage of computational automata theory.

ordered list of (x,y) cluster tuples where the first member of each tuple is a vowel or consonant cluster and the second is a position flag, "b", "m" or "e" which denotes whether the cluster appears at the beginning, in the middle or at the end of the original spelling. For example, the spelling "map" is preprocessed to produce "(m,b)(a,m)(p,e)" since "m" appears at the beginning, "a" in the middle and "p" at the end of the word. The results from preprocessing serves as input to a Sprees automaton, which operates in the following manner to recognize a preprocessed spelling.

A Sprees automaton is a deterministic finite state automaton defined by the quintuple:
$M_s = (Q, \Sigma, \delta, q_0, F)$, where

Q is a finite set of internal states,
Σ : {(x,y) | x is a consonant or vowel cluster and
y $\in$ {b, m, e} indicating x appears at the
beginning, in the middle, or at the end
of a spelling},
$\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
$q_0 \in Q$ is the start state,
$F \subseteq Q$ is a set of final states.

Definition 1. A Sprees automaton is a deterministic finite automaton.

At the initial time, the Sprees automaton is assumed to be at the start state q_0, reading the leftmost cluster tuple of the input spelling. During each move of the automaton, the input mechanism advances one tuple to the right, so each move consumes one cluster. When the end of the spelling is reached, the spelling is recognized if the automaton is in one of its final states; otherwise, the spelling is rejected.

Diagram 1 provides a visual representation of a Sprees automaton using a transition graph. The vertices represent states and the edges represent transitions based on clusters and their positions in the spelling. Each path from the start state to each final state composes a spelling. Traversing the graph from the start state to each final state using every arc, while writing down the clusters as they are encountered on each arc, provides a listing of all the spellings recognized by a Sprees automaton.

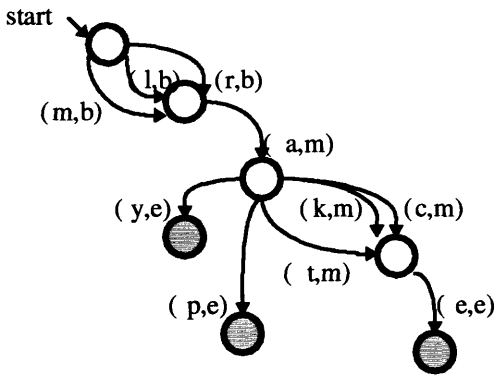

Diagram 1. A Sprees transition graph.
The transition graph above shows a Spree automaton formed from the training words "race," "late," "may," "lake" and "map." The shaded vertices are final states. There are 15 distinct paths through the graph representing the program's ability to recognize and generate 15 spellings.

If the Sprees automaton depicted in Diagram 1 also included the word "mat", then "rat" and "lat" would be additional spellings recognized by the machine. The first is an English word and the latter is not. This exemplifies the nature of generalization in the system. A Sprees automaton maintains a linear relationship between adjacent pairs of clusters. From Diagram 1, we see that "m" and "a" as well as "l" and "a" are adjacent clusters since there exists a sub-path that uses the "m" then the "a" arc and another sub-path that uses the "l" then the "a" arc. If the word "mat" was included in Diagram 1, then there would be a new arc labeled "(t,e)" that goes to a new final state. Since there is no restriction on getting to the new "t" arc by going first on the "m" arc, we

could use the "l" then "a" arcs. This gives a complete path that spells "lat." When a new cluster pair is incorporated into a Sprees automaton, it may add more than one path through the graph and these new paths are spellings that were not given to the automaton during training. These new spellings may be English words or nonsense words.

So, whenever a cluster appears in more than one training word, Sprees generalizes the linear relationship across all the spellings containing the cluster. At first glance this may seem all-inclusive but to understand its restrictions we must first consider how orthographic transitions are interpreted from the automaton. There are three different kinds of transitions as depicted in Diagrams 2a, 2b and 2c. One kind of transition is for clusters that begin a spelling, another for clusters in the middle of a spelling, and yet another transition for clusters at the end of a spelling.

The begin, middle and end spelling specifications limit where clusters can appear, making cluster placements consistent with witnessed locations in training words, and thus maintain systematic pronunciation relationships. For example, recall the word-initial b__ and word-final __mb pronunciations we discussed earlier. Another example is the cluster "ng." No English word begins ng__ but many end __ng due to the common suffix "ing."

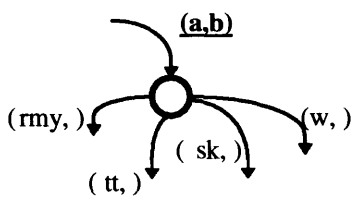

Diagram 2a. Transition sub-graph for clusters that begin a word.
The transition graph above shows how to read a transition for a cluster that begins a spelling: "a" at the beginning of a word is followed by: "rmy, tt, sk, or w."

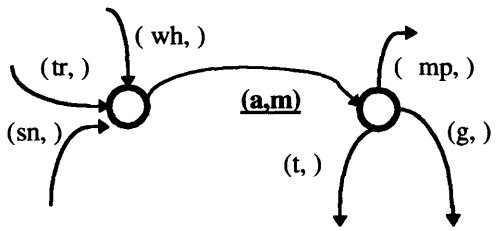

Diagram 2b. Transition sub-graph for clusters in the middle of a word.
The transition graph above shows how to read a transition for a cluster that appears in the middle of a spelling: "a" in the middle of a word is preceded by: "wh, sn, or tr;" and is followed by: "mp, g, or t."

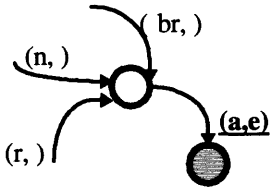

Diagram 2c. Transition sub-graph for clusters that end a word.
The transition graph above shows how to read a transition for a cluster that ends a spelling: "a" at the end of a word is preceded by: "br, r or n."

A Sprees automaton can be modified to include weights on each arc that reflect the frequency of each cluster's positional appearance during training. New spellings can then be generated that better reflect pronunciation patterns common to the training words. Since there is no "unlearning," the weights also provide a means for rating acquired knowledge.

Consider all the spellings recognized by a Sprees automaton. We consider the language of a Sprees automaton to be all the spellings it recognizes. Such languages are considered regular languages. Let L(E) be the set of commonly used English spellings. Let the Sprees automaton that recognizes the spellings of L(E) be M_s and the language of M_s be $L(M_s)$. Clearly $L(E) \subseteq L(M_s)$ and the difference, $L(M_s) - L(E)$ is the set of nonsense words. Later in the results section, we will provide some relative measurements for the sizes of these sets.

We have seen how a Sprees automaton recognizes spellings. Now we will look at how a Sprees automaton is constructed and then reflect on its generative power. A Sprees automaton is constructed using the algorithm presented as Algorithm 1.

Let M_s be the Sprees Automaton we are to build. M_s begins with only the start state.
Let T be the set of preprocessed training words from which we are to build a Sprees automaton.
For each word, $w \in T$, do the following:

a. Let the current state be the start state.
b. The tuples of w are ordered $c_1, c_2, \ldots, c_n$.
   Start with c_1 and proceed in order through c_n. For each c_i, do the following:
   - If c_i is already a transition from the current state, advance the current state to be the state after the transition and then process c_{i+1};
   - else if c_i is c_n then make a new final state for the transition c_i, and then process the next w;
   - else if there exists a transition anywhere in M_s based on c_{i+1} , then make a transition from the current state to the state from which c_{i+1} is the source and advance the current state be the state at the c_{i+1} transition, and then, process c_{i+1};
   - else make a new state based on the transition c_i and then advance the current state be the newly created state, and then, process c_{i+1}.

Algorithm 1. Constructs a Sprees Automaton from training words.

The language of a Sprees automaton is infinite whenever the automaton contains a cycle and from Algorithm 1 this happens whenever a cluster repeats within a spelling. Consider the Sprees automaton in Diagram 3 that recognizes the spelling "gigabit" where the "i" cluster repeats. The recognized spellings are {"git", "gigabit", "gigabigabit", "gigabigabigabit", ...}. If any one spelling contains an inner cluster that repeats, the language of the Sprees automaton that recognizes that spelling is infinite. This is an orthographic parallel to the infinite scope of language found at the syntactic level. We could never exhaustively list all the sentences of a natural language, because no matter how many sentences appear on the list, we could always

produce a longer one. Further, since the semantics of traversing adjacent transitions in a Sprees graph involve the ability to pronounce the result, the compositional property found at the syntactic level is present here also. All spellings produced by Sprees, including the longer ones, are pronounceable consistent with the pronunciation found in the original training words.

Recall our earlier discussion of bigram-like sequences which recorded the order in which two adjacent clusters appeared. Every two arcs on an automaton that share a common vertex form a bigram-like sequence and we consider their arcs to be adjacent. Diagram 3 contains five sequences including "i in the middle of a word is followed by g" and "t at the end of a word is preceded by i." Care is taken so that each pair of adjacent arcs count as only one sequence.

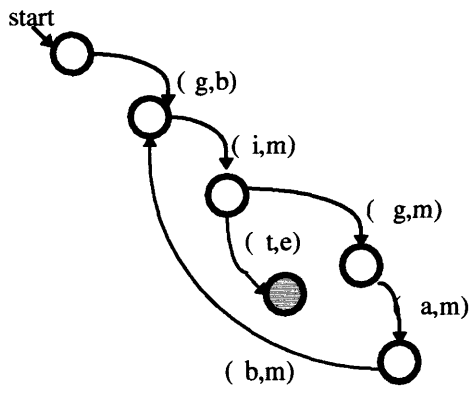

Diagram 3. Transition graph with a cycle.
The transition graph above shows a Spree automaton formed from the training word "gigabit." The language recognized by this automaton is infinite.

Beyond the number of spellings an automaton can recognize lies its generative power, which is based on the set of languages it can represent. The grammar of Sprees is described in Definition 2. It is a regular grammar that is right-linear and right-linear grammars generate regular

languages, which we already know is the language of a Sprees automaton. Regular grammars are the most restricted class of Chomsky's grammatical formalisms. Their generative capacity is considered the least powerful (Chomsky, 1957). The generalizations possible in a Sprees automaton are of the form *a*b** which is a sequence of any number of *a* followed by any number of *b*. Knowing this helps explain further limitations on how spellings are generalized by a Sprees automaton. It is actually a least-general approach.

The grammar of a Sprees automaton G_s is defined as a quadruple, $G_s = (V, T, S, P)$, where

V is a finite set of objects called variables,
T is a finite set of objects called clusters,
$S \in V$ is a special symbol called the start variable,
P is a finite set of productions such that all
productions are of the form:
$A \rightarrow xB$,
$A \rightarrow x$, where $A, B \in V$, and $x \in T^*$.

V and T are non-empty and disjoint.

Definition 2. The grammar of a Sprees automaton is a regular grammar.

Generalization in Sprees is tightly bounded. Clusters and the transitions from cluster to cluster are limited to those actually experienced in the training words and are not decided on beforehand. English words have alternating vowel and consonant clusters and a Sprees automaton maintains this invariant by using vowel and consonant clusters. The begin, middle and end spelling rules limit where clusters can appear in a spelling and this preserves systematic pronunciation relationships such as word-initial and word-final pronunciation. On the other hand, there are three forms of generalization. The first happens when a cluster appears in more than one word; the second happens when a cluster recurs in the middle of a spelling; and, the third involves

allowing transitions to clusters at the beginning of a word when the cluster actually appears in the middle of a training word. In this last case, rather than the transitions requiring consistent positions within words (such as beginning, middle or end), the beginning and middle positions collapse into one position. Word final positioning is still maintained; the end result allows generalizations to include word concatenations. Since the generalization remains restrictive to positioning and clusters actually observed, we term this a least-general approach.

As for computational time, a Sprees automaton can recognize a spelling in optimal O(n) time, where n is the number of clusters in the spelling. This is small in English and as a result Sprees is quite fast. A graph traversal algorithm can be modified to print regular expressions for all spellings generated by a Sprees automaton. Regular expressions use parenthesis and asterisks to denote which characters can recur. Generating all possible spelling paths takes $O(c^2)$ time where c is the number of cluster tuples in the graph. Storage requirements for a Sprees automaton is also $O(c^2)$. So recognizing a given spelling is very fast and generating a single word is also fast since it is a random walk on the graph.

4 Results

We ran the Sprees program on three dictionary word lists. The first dictionary list was from Macmillan Very First Dictionary (1983) and contains words for children ages 5-8. The second dictionary list was included with the Unix ISPELL program, and the third dictionary list was from Webster's Unabridged Dictionary (1983). Sprees automata were built for each of these lists. The results are listed in Table 5. Notice that the number of words increase an order of magnitude with

each list yet the clusters do not grow rapidly. We certainly expect the number of cluster tuples and bigram-like sequences to converge since given the immense number of theoretically possible spellings, only a small number actually occur, and the majority of these recur in patterns common to many words. See Table 6 for a relative plot of these values. Notice how a ratio of two sequences per cluster tuple is constant, independent of the size of the list. This means that most cluster tuples are not included in a begin, middle and an end sequence. They are only in about two of them.

Source	#Words	#Clusters.	#Bigram-like sequences
1. Children's Dictionary	1,408	249	551
2. Spell Checker Dictionary	25,145	1633	3383
3. Webster's Unabridged	213,556	4200	8715

Table 5. Size measurements for different Sprees automata.
The number of cluster tuples and bigram-like sequences are counted after building a Sprees automaton using only the indicated source list. On the transition graph, a bigram-like sequence reflects the transition between two arcs incident to a common vertex.

We have already seen that the number of spellings for a Sprees automaton is infinite whenever a cluster repeats in a single spelling. Yet we will provide some measure of the number of nonsense words. We can detect cycles in the automaton and we can count the number of distinct paths by carefully not repeating cycles. This provides a measure from which we can subtract the number of legal spellings and get the number of nonsense words. This number was so large it was not tractable on the dictionary lists we used earlier. Instead, we used tiny lists; see Appendix A for a complete list training words and the spellings they generated. Table 7 provides a summary.

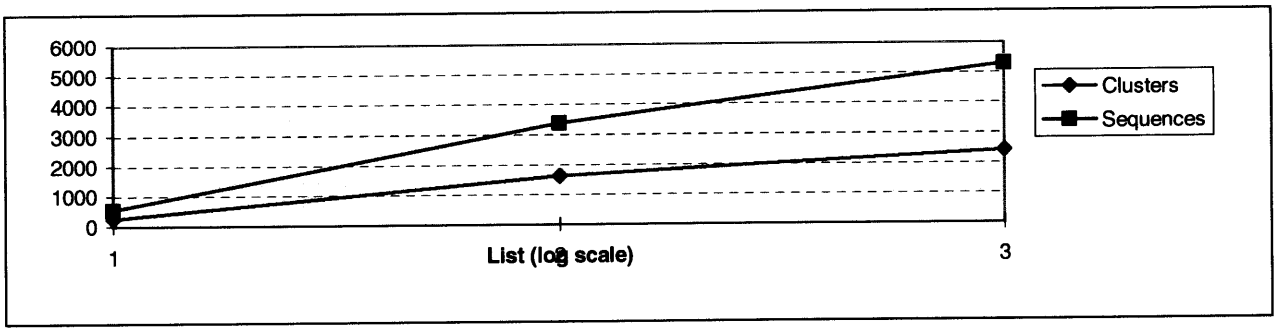

Table 6. Plot of size measurements for different Sprees automata.
Based on the lists in Table 5.

Clearly, the number of nonsense words is tremendous. In the next section we will report on an experiment where subjects classify some new words created by a Sprees automaton. The objective is to determine how much of the sound that characterizes the training set remains in the newly created words.

Source	#Words	#Clusters	#Bigram-like sequences	#Generate
female names	12	18	43	346
family names	12	26	48	232

Table 7. Number of newly created spellings.
The number of different spellings a Sprees automaton can generate and recognize given a small number of the training words. The actual number of words the Sprees automaton can generate is infinite, so spellings containing cycles are counted only once.

4.1 Experiment: Humans Categorize New Words

We conducted an experiment to determine how humans would categorize new words generated by the Sprees program. The subjects were 6 adults, all native English speakers. Each of the

adults were given five (5) printed pages. Each page had a list of about 20 words on top and the bottom of the page had five boxes. The subjects were to mark a box depending on whether the words sounded more like: (a) female first names, (b) male first names, (c) family names, or (d) medical terms. There was a box for each of these options and an additional box (e) to select if they could not decide. Appendix B contains samples of these pages.

The words that appeared at the top of each page were generated by the Sprees program using the following training words: (1) a list of 12 common female names; (2) a list of 15 common male names; (3) a list of 15 common family names; (4) a list of 15 common medical terms; and, (5) a list that contained all 60 spellings from (1) through (4). Appendix C contains a list of these training words. Each subject received samples of about 20 nonsense words generated by the Sprees program after it trained on each of these six training sets. We generated a separate list from each training set. The subjects matched the lists almost perfectly to their source. See Table 8 for a summary.

A perfect match by the subjects would have all 6's along the diagonal. Though the results are not perfect, they clearly show that new words created by Sprees maintain the "sound" of the original training words from which those spellings were generated.

	Sources					
	(1)	(2)	(3)	(4)	(5)	errors
(a)	6					0
(b)		5	1			1
(c)			5		1	1
(d)				6		0
(e)					6	0
errors		0	1		1	2

Table 8. Results of human experiment.
Each column represents the source that generated the list and each row represents the box that was marked by the subjects. The values for (1)-(5) and (a)-(e) are presented in the text.

5 Discussion

In concluding, we will discuss some of the phonological issues surrounding Sprees and its ability to capture pronunciation in spelling. Then, we will contrast and compare the learning of automata using the Sprees algorithm with algorithms found in machine learning.

5.1 Phonemes and Graphemes.

There is a special relationship between the phonology and orthography of the English language. The letters in the English alphabet correspond roughly to phonemes, which are the basic speech sounds of a language. Rough is the operant word here. Some linguists may view English orthography as a faulty phonetic system and consequently believe that much of the graphemic patterning in the orthography should be ignored.

It is true that the phonemes of a language are not determined by the letters we write but by the sounds we speak. In this sense, vowels are not "a, e, i, o, u and sometimes y," but are those sounds in which the stream of air coming from the lungs passes through the mouth and nose with no audible friction. Vowel sounds may further be classified according to the position in which the tongue is held in the mouth during their pronunciation. Chomsky and Halle (1968) provide a complete set of features that describe English phonemes. Under the phonetic feature set, there are more than eleven vowel phonemes in English.

There has been research that maps graphemes to phonemes. Robert Hall (1961) did so by hand in his work on sound and spelling in English. A grapheme is a sequence of letters that appear in a spelling and phonologically operate as a single unit like "ch" and "ed." A Sprees automaton does not break a spelling into the same grapheme units used by Hall; instead Sprees uses vowel and consonant clusters as the atomic units. The grapheme "ed" for example would be two adjacent clusters in a Sprees automaton but is usually considered a single grapheme, and in the spelling "stopped," the grapheme "ed" is pronounced as a single phoneme /t/. Table 9 maps some sample phonemes to graphemes. In closing, we reflect on how people recognize new spellings.

Sound	Phoneme	Grapheme	Samples
b*ea*t	/i/	ee	meet
		ea	sea
		i	machine
b*i*t	/I/	i	hit
		o	women
		y	myth
h*o*t	/ɑ/	a	father
		ea	heart
		e	sergeant
*t*igh*t*	/t/	ed	stopped
		t	hunter

Table 9. Phoneme to grapheme mapping.
Some sample sounds mapped to their corresponding phoneme and some example graphemes.

5.2 Humans Recognize and Generate Words

Learning to read a foreign language differs radically from learning to read one's native language. In learning to read a foreign language, we often have no prior knowledge of the language and so we tend to translate directly from writing to meaning. Learning to read our native language is different. Usually we already have the ability to speak the language, so reading in this situation requires primarily translating written symbols to sound.

In the sense of Sprees, we are a good reader if we cannot only pronounce all the words which we were taught to read, but also if we can pronounce a high percentage of new words we encounter. Certainly if we cannot pronounce words we have not seen before we are a deficient reader. Good readers therefore must form some spelling-to-sound generalizations. What these generalizations are, how they develop, and how they differ among people has been the topic of many experimental psychology studies. We do not assert that Sprees is a model of how people

generalize spellings, only that like people, Sprees generalizes spellings consistent with sound-to-spelling correspondences previously learned.

5.3 Learning Automata

Recall Algorithm 1 presented earlier which constructs a Sprees automaton from training words. There are several algorithms in the field of machine learning that also "learn" by building a deterministic finite state automaton, in particular consider Probably Approximately Correct or PAC learning (Valiant, 1984) and Angluin's L* algorithm (1987). There are distinctive differences in the goals and assumptions made in these learning environments from those made in the Sprees environment. First, the teacher in the machine learning environment typically provides negative as well as positive examples. This is not done in the Sprees environment. Instead, all training words are positive examples of membership.

Second, Algorithm 1 does not infer beyond the transitions actually witnessed; therefore, it has no internal hypotheses nor does it pose membership questions. The learning is passive and rote. The L* algorithm, on the other hand, gets more information than just a list of examples; it utilizes a "teacher" that answers membership questions ("is *s* acceptable?") and equivalence queries ("are automata A and B equivalent?"). The algorithm poses experiments to distinguish the states. Its goal is to get the DFA exactly right, while PAC learning gets the DFA correct within some given error bound. In Sprees, the notion of an exact solution is quite different; the exact solution to Sprees is the one directly deduced from its training words.

Third, the L* algorithm requires knowledge of the complete alphabet beforehand and the alphabet is not expected to change. In the Sprees environment, new clusters can be encountered after, as well as, during the learning process. Suppose we want to build the automaton that recognizes only the 15 words in Table 4. The transition graph for the resulting Sprees automaton is in Diagram 1; it was built by providing Algorithm 1 with 5 example words which are also included in Table 4. The result of the L* algorithm produces the same transition graph as Algorithm 1, but the complete cluster alphabet had to be provided beforehand and a teacher had to answer queries that required complete knowledge of all acceptable spellings. The L* algorithm and Sprees both learn finite automata, but the Sprees Algorithm 1 generalizes its observations while the L* algorithm can offer solutions to more refined problems. However, the Sprees algorithm is more efficient at its specified task since its alphabet can be determined during operation. It also spends no time trying to unearth an "exact" solution since its specified goal is not usually based on a known set of spellings beforehand but actually involves a discovery process for the algorithm as well as the "teacher".

In concluding, we have presented a system that captures phonological information found in training words and can be used to generate words that sound similar to its training words and to recognize words that "sound" as though they belong in the training words even though they were not part of the original training set. The system was used to recognize personally-identifying information in unrestricted text and to replace such words with made-up alternatives. The power of the system resides in its representation of words as a sequence of vowel and consonant clusters and its ways of generalizing the relationships between clusters.

References

American Heritage Dictionary of the English Language. New York: Houghton Mifflin Company. 1970.

Angluin, D., Learning Regular Sets from Queries and Counterexample. *Information and Computation.* 1987:(75):87-106.

Bird, E. and Ellison, T., One-level Phonology: Autosegemental Representations and Rules as Finite Automata *Computational Linguistics.* 1994:20(1):55-90.

Carbonell, J. and Hayes, P., Recovery Strategies for Parsing Extragrammatical Language. *Computational Linguistics.* 1983:9(3-4):123-146.

Chomsky, N., *Syntactic structures.* Mouton: The Hague and Paris. 1957.

Chomsky, N. and Halle, M., *The sound pattern of English.* Cambridge: The MIT Press. 1968.

Durham, I., Lamb, D. and Saxe, J., Spelling Correction in User Interfaces. *Communications of the ACM.* 1983:26.

Gross, M., The use of finite automata in the lexical representation of natural language. *Electronic Dictionaries and Automata in Computational Linguistics. LITP Spring School on Theoretical Computer Science, Proceedings.* Berlin. Springer-Verlag. 1989

Hall, R., *Sound and spelling in English.* New York: Chilton Books. 1961.

Hunnicutt, S., Phonological Rules for a Text-to-Speech System. *American Journal of Computational Linguistics.* 1976.

Macmillan Very First Dictionary. New York Macmillan Publishing Company 1983.

Nilsen, A., Why Big Businesses Break Spelling Rules*English Journal* 1994;83(5):48-53.

Roche, E. and Schabes, Y., Deterministic Part-of-Speech Tagging with Finite-State Transducers. *Computational Linguistics.* 1995:21(2).

Roche, E. and Schabes, Y. *Finite-State Language Processing.* Cambridge. MIT Press. 1997.

Sipser, M., *Introduction to the Theory of Computation.* New York: PWS Publishers. 1996.

Sweeney, L., Replacing Personally-Identifying Information in Medical Records, the Scrub System. In: Cimino, JJ, ed. Proceedings, *Journal of the American Medical Informatics Association.* Washington, DC: Hanley & Belfus, Inc, 1996:333-337.

Valiant, L., A Theory of the Learnable *Communications of the ACM.* 1984:27(11):1134-1142.

Venezky, R., *A computer program for deriving spelling to sound correlations*, Masters thesis, Cornell University, 1962. Published in part in Harry Levin, et.al. *A basic research program on reading*. Ithaca, 1963.

Webster's Unabridged Dictionary. New York: Simon and Schuster. 1983.

Appendix A

Below are 12 female names used to build a Sprees automaton and a list of the 346 spellings the automaton generated. Following that is a list of 12 family names used to build another Sprees automaton and a list of the 232 spellings that automaton generated. Of course, the actual number of spellings these automata can generate is infinite, so spellings containing cycles are counted only once and the recurring characters are printed within parenthesis.

12 Female Names for Training:

Nicole
Lauren
Heather
Laura
Rachel
Amy
Erin
Sara
Amber
Kimberly
Katherine
Lisa

346 Female Names Generated:

amberly
ambel
amber
amberisa
amberisatherly
amberisathel
amberisather
ambe(risathe)*
amberisathen
amber(isar)*...
ambera
amberatherly
amberathel
amberather
ambe(rathe)*
amberathen
amberarisa
amberarisatherly
amberarisathel
amberarisather
ambe(rarisathe)*
amberarisathen
ambera(risa)*...
amberacherly
amberachel
amberacher
ambe(rache)*
amberachen
ambererly
amberel
amberer
ambe(re)*
amberen
amben
amy
er
erisa
erisatherly
erisathel
erisather
er(isather)*...
era
eratherly
erathel
erather
eratherisa
era(therisa)*...
erarisa
erarisatherly
erarisathel
erarisather
erar(isather)*...
erara
er(ar)*
erarerly
erarel
erarer
erarerisa
erarerisatherly
erarerisathel
erarerisather
erare(risathe)*
erarerisathen
era(rerisa)*...
eracherly
erachel
eracher
eracherisa
eracherisatherly
eracherisathel
eracherisather
erache(risathe)*
eracherisathen
eracher(isar)*...
erachera
er(acher)*
erachererly
eracherel

eracherer
erache(re)*
eracheren
erachen
ererly
erel
erer
ererisa
ererisatherly
ererisathel
ererisather
ere(risathe)*
ererisathen
erer(isar)*...
erera
ereratherly
ererathel
ererather
ere(rathe)*
ererathen
ererarisa
ererarisatherly
ererarisathel
ererarisather
ere(rarisathe)*
ererarisathen
erera(risa)*...
ereracherly
ererachel
ereracher
ere(rache)*
ererachen
er(er)*...
heatherly
heathel
heather
heatherisa
heath(erisath)*...
katherly
kathel
kather
katherisa
ka(therisa)*...
karisa
karisatherly
karisathel
karisather
kar(isather)*...
kara
karatherly
karathel
karather
karatherisa
kara(therisa)*...
ka(ra)*...
kacherly
kachel
kacher
kacherisa
kacherisatherly
kacherisathel
kacherisather
kache(risathe)*
kacherisathen
kacher(isar)*...
kachera
kacheratherly
kacherathel
kacherather
kache(rathe)*
kacherathen
kacherarisa
kacherarisatherly
kacherarisathel
kacherarisather
kache(rarisathe)*
kacherarisathen
kachera(risa)*...
ka(chera)*...
kisa
kisatherly
kisathel
kisather
ki(satheri)*...
kimberly
kimbel
kimber
kimberisa
kimberisatherly
kimberisathel
kimberisather
kimbe(risathe)*
kimberisathen
kimber(isar)*...
kimbera
kimberatherly
kimberathel
kimberather
kimbe(rathe)*
kimberathen
kimberarisa
kimberarisatherly
kimberarisathel
kimberarisather
kimbe(rarisathe)*
kimberarisathen
kimbera(risa)*...
kimberacherly
kimberachel
kimberacher
kimbe(rache)*
kimberachen
kimbererly
kimberel
kimberer
kimbe(re)*
kimberen
kimben
kin
kine
kicole
lisa
lisatherly
lisathel
lisather
li(satheri)*...
limberly
limbel
limber
limberisa
limberisatherly
limberisathel
limberisather
limbe(risathe)*
limberisathen
limber(isar)*...
limbera
limberatherly
limberathel
limberather
limbe(rathe)*
limberathen
limberarisa
limberarisatherly
limberarisathel
limberarisather
limbe(rarisathe)*
limberarisathen
limbera(risa)*...
limberacherly
limberachel
limberacher
limbe(rache)*
limberachen
limbererly
limberel
limberer
limbe(re)*
limberen
limben
lin
line
licole
laurisa
laurisatherly
laurisathel
laurisather
laur(isather)*...
laura
lauratherly
laurathel
laurather
lauratherisa
laura(therisa)*...
laurarisa
laurarisatherly
laurarisathel

laurarisather
laurar(isather)*...
laurara
laur(ar)*
laurarerly
laurarel
laurarer
laurarerisa
laurarerisatherly
laurarerisathel
laurarerisather
laurare(risathe)*
laurarerisathen
laura(rerisa)*...
lauracherly
laurachel
lauracher
lauracherisa
lauracherisatherly
lauracherisathel
lauracherisather
laurache(risathe)*
lauracherisathen
lauracher(isar)*...
laurachera
laur(acher)*
laurachererly
lauracherel
lauracherer
laurache(re)*
lauracheren
laurachen
laurerly
laurel
laurer
laurerisa
laurerisatherly
laurerisathel
laurerisather
laure(risathe)*
laurerisathen
laurer(isar)*...
laurera
laureratherly
laurerathel
laurerather
laure(rathe)*
laurerathen
laurerarisa
laurerarisatherly
laurerarisathel
laurerarisather
laure(rarisathe)*
laurerarisathen
laurera(risa)*...
laureracherly
laurerachel
laureracher
laure(rache)*
laurerachen
laur(er)*...
nisa
nisatherly
nisathel
nisather
ni(satheri)*...
nimberly
nimbel
nimber
nimberisa
nimberisatherly
nimberisathel
nimberisather
nimbe(risathe)*
nimberisathen
nimber(isar)*...
nimbera
nimberatherly
nimberathel
nimberather
nimbe(rathe)*
nimberathen
nimberarisa
nimberarisatherly
nimberarisathel
nimberarisather
nimbe(rarisathe)*
nimberarisathen
nimbera(risa)*...
nimberacherly
nimberachel
nimberacher
nimbe(rache)*
nimberachen
nimbererly
nimberel
nimberer
nimbe(re)*
nimberen
nimben
nin
nine
nicole
ra
ratherly
rathel
rather
ratherisa
ra(therisa)*...
rarisa
rarisatherly
rarisathel
rarisather
rar(isather)*...
rara
raratherly
rarathel
rarather
raratherisa
rara(therisa)*...
ra(ra)*...
racherly
rachel
racher
racherisa
racherisatherly
racherisathel
racherisather
rache(risathe)*
racherisathen
racher(isar)*...
rachera
racheratherly
racherathel
racherather
rache(rathe)*
racherathen
racherarisa
racherarisatherly
racherarisathel
racherarisather
rache(rarisathe)*
racherarisathen
rachera(risa)*...
ra(chera)*...
sa
satherly
sathel
sather
satherisa
sa(therisa)*...
sarisa
sarisatherly
sarisathel
sarisather
sar(isather)*...
sara
saratherly
sarathel
sarather
saratherisa
sara(therisa)*...
sa(ra)*...
sacherly
sachel
sacher
sacherisa
sacherisatherly
sacherisathel
sacherisather
sache(risathe)*
sacherisathen
sacher(isar)*...
sachera
sacheratherly
sacherathel
sacherather

sache(rathe)*
sacherathen
sacherarisa
sacherarisatherly
sacherarisathel
sacherarisather
sache(rarisathe)*
sacherarisathen
sachera(risa)*...
sa(chera)*...

---

12 Family Names for Training:

Smith
Johnson
Williams
Brown
Jones
Miller
Davis
Wilson
Anderson
Taylor
Moore
Thomas

232 Family Names Generated:

andersomas
anderso(maylo)*...
andersor
andersown
anderson
ande(rsone)*...
ander
andes
bromas
bro(maylo)*...
bror
brown
bron
bronersomas
bronerso(maylo)*...
bronersor
bronersown
bronerson
bro(nerso)*
bronersohnsomas
bronersohnso(maylo)*...
bronersohnsor
bronersohnsown
bronersohnson
bro(nersohnso)*
bronerso(hnso)*...
broner
brones
brohnsomas
brohnso(maylo)*...
brohnsor
brohnsown
brohnson
brohnsonersomas
brohnsonerso(maylo)*...
brohnsonersor
brohnsonersown
brohnsonerson
brohnso(nerso)*
bro(hnsonerso)*...
das
daylomas
da(yloma)*...
davilsomas
davilso(maylo)*...
davilsor
davilsown
davilson
davilsonersomas
davilsonerso(maylo)*...
davilsonersor
davilsonersown
davilsonerson
davilso(nerso)*
davilsonersohnsomas
davilsonersohnso(maylo)*...
davilsonersohnsor
davilsonersohnsown
davilsonersohnson
davilso(nersohnso)*
davilsonerso(hnso)*...
davilsoner
davilsones
davilsohnsomas
davilsohnso(maylo)*...
davilsohnsor
davilsohnsown
davilsohnson
davilsohnsonersomas
davilsohnsonerso(maylo)*...
davilsohnsonersor
davilsohnsonersown
davilsohnsonerson
davilsohnso(nerso)*
davilso(hnsonerso)*...
davis
davillersomas
davillerso(maylo)*...
davillersor
davillersown
davillerson
daville(rsone)*...
daviller
davilles
davilliams
davith
jomas
jo(maylo)*...
jor
jown
jon
jonersomas
jonerso(maylo)*...

jonersor
jonersown
jonerson
jo(nerso)*
jonersohnsomas
jonersohnso(maylo)*...
jonersohnsor
jonersohnsown
jonersohnson
jo(nersohnso)*
jonerso(hnso)*...
joner
jones
johnsomas
johnso(maylo)*...
johnsor
johnsown
johnson
johnsonersomas
johnsonerso(maylo)*...
johnsonersor
johnsonersown
johnsonerson
johnso(nerso)*
jo(hnsonerso)*...
moore
milsomas
milso(maylo)*...
milsor
milsown
milson
milsonersomas
milsonerso(maylo)*...
milsonersor
milsonersown
milsonerson
milso(nerso)*
milsonersohnsomas
milsonersohnso(maylo)*...
milsonersohnsor
milsonersohnsown
milsonersohnson
milso(nersohnso)*
milsonerso(hnso)*...
milsoner
milsones
milsohnsomas
milsohnso(maylo)*...
milsohnsor
milsohnsown
milsohnson
milsohnsonersomas
milsohnsonerso(maylo)*...
milsohnsonersor
milsohnsonersown
milsohnsonerson
milsohnso(nerso)*
milso(hnsonerso)*...
mis
millersomas
millerso(maylo)*...
millersor
millersown
millerson
mille(rsone)*...
miller
milles
milliams
mith
smilsomas
smilso(maylo)*...
smilsor
smilsown
smilson
smilsonersomas
smilsonerso(maylo)*...
smilsonersor
smilsonersown
smilsonerson
smilso(nerso)*
smilsonersohnsomas
smilsonersohnso(maylo)*...
smilsonersohnsor
smilsonersohnsown
smilsonersohnson
smilso(nersohnso)*
smilsonerso(hnso)*...
smilsoner
smilsones
smilsohnsomas
smilsohnso(maylo)*...
smilsohnsor
smilsohnsown
smilsohnson
smilsohnsonersomas
smilsohnsonerso(maylo)*...
smilsohnsonersor
smilsohnsonersown
smilsohnsonerson
smilsohnso(nerso)*
smilso(hnsonerso)*...
smis
smillersomas
smillerso(maylo)*...
smillersor
smillersown
smillerson
smille(rsone)*...
smiller
smilles
smilliams
smith
tas
taylomas
ta(yloma)*...
tavilsomas
tavilso(maylo)*...
tavilsor
tavilsown
tavilson
tavilsonersomas
tavilsonerso(maylo)*...
tavilsonersor
tavilsonersown
tavilsonerson
tavilso(nerso)*
tavilsonersohnsomas
tavilsonersohnso(maylo)*...
tavilsonersohnsor
tavilsonersohnsown
tavilsonersohnson
tavilso(nersohnso)*
tavilsonerso(hnso)*...
tavilsoner
tavilsones
tavilsohnsomas
tavilsohnso(maylo)*...
tavilsohnsor
tavilsohnsown
tavilsohnson
tavilsohnsonersomas
tavilsohnsonerso(maylo)*...
tavilsohnsonersor
tavilsohnsonersown
tavilsohnsonerson
tavilsohnso(nerso)*
tavilso(hnsonerso)*...
tavis
tavillersomas
tavillerso(maylo)*...
tavillersor
tavillersown
tavillerson
taville(rsone)*...
taviller
tavilles
tavilliams
tavith
thomas
tho(maylo)*...
thor
thown
thon
thonersomas
thonerso(maylo)*...
thonersor
thonersown
thonerson
tho(nerso)*
thonersohnsomas
thonersohnso(maylo)*...
thonersohnsor
thonersohnsown
thonersohnson
tho(nersohnso)*
thonerso(hnso)*...
thoner

thones
thohnsomas
thohnso(maylo)*...
thohnsor
thohnsown
thohnson
thohnsonersomas
thohnsonerso(maylo)*...
thohnsonersor
thohnsonersown
thohnsonerson
thohnso(nerso)*
tho(hnsonerso)*...
wilsomas
wilso(maylo)*...
wilsor
wilsown
wilson
wilsonersomas
wilsonerso(maylo)*...
wilsonersor
wilsonersown
wilsonerson
wilso(nerso)*
wilsonersohnsomas
wilsonersohnso(maylo)*...
wilsonersohnsor
wilsonersohnsown
wilsonersohnson
wilso(nersohnso)*
wilsonerso(hnso)*...
wilsoner
wilsones
wilsohnsomas
wilsohnso(maylo)*...
wilsohnsor
wilsohnsown
wilsohnson
wilsohnsonersomas
wilsohnsonerso(maylo)*...
wilsohnsonersor
wilsohnsonersown
wilsohnsonerson
wilsohnso(nerso)*
wilso(hnsonerso)*...
wis
willersomas
willerso(maylo)*...
willersor
willersown
willerson
wille(rsone)*...
willer
willes
williams
with

Appendix B

Below are copies of five (5) lists of spellings that humans were asked to categorize. Each list of spellings was generated from a Sprees automata that learned on a list of either female or male first names, family names, or medical terms. One list was generated from the combination of all these lists. See Appendix C for a list of words on which the Sprees automata were built. The order in which the word lists appear in this Appendix are female names, male names, family names, medical terms and random spellings.

Look at the list of words below. Most likely they are words you've never seen before. Check the box at the bottom that best classifies these words. Select only one box.

amberly
ambel
amberisa
ambera
erisa
era
erel
heathel
kathel
karisa
kara
kachera
kisa
kimbel
kimber
kimbera
licole
laurisa
nisa
ratherly
rachera
satherly

a. female first names
b. male first names
c. family names
d. medical terms
e. cannot decide

Look at the list of words below. Most likely they are words you've never seen before. Check the box at the bottom that best classifies these words. Select only one box.

anthon
bennis
bip
billis
bratthew
coben
coris
corillis
chrillip
dandon
dert
dennill
dobert
dony
dorey
doric
mandon
phid
robey
roric

a. female first names

b. male first names

c. family names

d. medical terms

e. cannot decide

Look at the list of words below. Most likely they are words you've never seen before. Check the box at the bottom that best classifies these words. Select only one box.

broner
davilson
jonerson
milsor
milson
millerson
milliams
smilsoner
smilsones
smis
tavilsown
tavilson
tavis
taviller
thones
wilsor
wilsown
willerson
willer
willes

a. female first names

b. male first names

c. family names

d. medical terms

e. cannot decide

Look at the list of words below. Most likely they are words you've never seen before. Check the box at the bottom that best classifies these words. Select only one box.

arthris
arthric
diseadiac
arthrisease
cardiatric
catorrhage
diseadiatric
hedage
hemorrhache
herve
tulmonage
hespitis
headardiatric
headage
hemonary
pediac
cancepage
pulmor
syndromorrhache
syndromonary

a. female first names

b. male first names

c. family names

d. medical terms

e. cannot decide

Look at the list of words below. Most likely they are words you've never seen before. Check the box at the bottom that best classifies these words. Select only one box.

coric
chrill
erid
dob
coren
ditic
ner
perve
reme
pedage
erel
kine
lin
amben
ererly
bron
mith
tas
thones
wis

a. female first names

b. male first names

c. family names

d. medical terms

e. cannot decide

Appendix C

Below are lists of spellings on which Sprees automata were built. A subset of nonsense words were then generated from each automaton; see Appendix B. Human subjects were asked to categorize the nonsense words in order to determine whether the nonsense words resembled their source.

Female Names:

nicole
lauren
heather
laura
rachel
amy
erin
sara
amber
kimberly
katherine
lisa
jane
susan
sylvia

Family Names:

smith
johnson
williams
brown
jones
miller
davis
wilson
anderson
taylor
moore
thomas
robinson
nelson
stevens

Male Names:

bill
phillip
eric
corey
don
dennis
ben
matthew
brandon
anthony
david
mark
chris
robert
joe

Medical terms:

cancer
arthritis
respiratory
disease
nerve
pulse
headache
syndrome
hemorrhage
cardiac
pediatric
bandage
tumor
hepatitis
pulmonary