Learning English Pronunciation Rules:
A Machine Learning Approach

Jian Zhang and Howard J. Hamilton

Dept. of Computer Science
University of Regina
Regina, Saskatchewan, Canada, S4S 0A2
jian,hamilton@cs.uregina.ca
phone: (306) 585-4977 fax: (306) 585-4745

Abstract

This paper presents LEP, a software system for Learning English Pronunciation. LEP contains four learning components. We emphasize one of the learning components called LE-PG, Learning English Pronunciation for Graphemes, which has been implemented and tested. When applied to the task of learning one-syllable words from an on-line pronouncing dictionary, LE-PG correctly predicted the pronunciation of 98.79% of the grapheme and 95.65% of the word. Apply the rules learned by LE-PG to one-syllable words, we found that the cumulative frequencies of the rule usage follows the Bradford-Zipf distribution.

1 Introduction

As English plays an increasingly important role in scientific research, technological development, and communication among nations in the world, it is gradually becoming the most accepted world language. Although English has many advantages as the world language, the present English orthography is “not merely a letter-to-sound system” [17] [10]. The smallest written forms corresponding to the basic sound units are not solely individual letters as most systematic or phonemic orthographies should have. Instead, the smallest written forms that correspond to their sounds are graphemes, sometimes single letters, sometimes two or even three letters. For example, tough has five letters which correspond to 3 phonemes. Theses 3 phonemes divide the word into 3 graphemes: <t>, <ou>, and <gh>, where each grapheme corresponds to one sound. But the grapheme <gh> in through is not pronounced and we call it a silent grapheme [20] [24].

Because the present English orthography is not a letter-to-sound system, it makes learning difficult. In this paper, we present a machine-learning system called LEP (Learning English Pronunciation),
which learns English pronunciation rules from the NTC2 [24] on-line pronouncing dictionary. In particular, we describe the LE-PG (Learning English Pronunciation for Graphemes) system. LE-PG is one of the four major learning components of LEP learning system [20], [21]. It has been implemented and tested on one-syllable words. The experimental results show 95.65% accuracy when applied to unseen words.

The remainder of the paper is divided into four sections. Section 2 introduces the LEP learning system and its four major components. It also includes the LEP system model, the logical units, the dataflow, and the software structure of LEP. Section 3 describes the LE-PG component, explains how LE-PG learns the pronunciation rules from a series of instances specified in the International Phonetic Alphabet (IPA) [14], and shows some interesting results. In Section 4, we present the design of other three learning components, LE-GS (Learning English Grapheme Segmentation), LE-SR (Learning English Syllabification Rules), and LE-SS (Learning English Stress for Syllables). Finally, Section 5 summarizes the LEP learning system and gives future research directions.

2 Overview of LEP

LEP learns English pronunciation rules by repeatedly selecting example words from the NTC2 on-line pronouncing dictionary. Different types of examples are generated from these words according to the needs of particular learning tasks. Each task has a target concept to learn, such as “grapheme”, “syllable”, “pronunciation”, or “stress”. To learn these concepts, LEP must evaluate many positive and negative examples to discover conditions that are true for the positive examples and false for the negative examples. The conditions can be complicated and many of them are contradictory.

LEP can be seen as an instance of Cohen and Feigenbaum’s model of a learning system (Cohen & Feigenbaum 1982, p.327) as shown in Figure 1. The names of the components of LEP are listed in Table 1. As a learning system, LEP contains the environment, the learning element, the knowledge base, and the performance element. The arrows in the diagram show the direction of data flow. The environment supplies information about graphemes, syllables, stresses, and pronunciations to the learning element. Each learning component in the learning element takes a different set of information and learns from it. All learned knowledge is stored in an explicit knowledge base. The performance element uses the knowledge learned to perform its task and provide feedback information to the learning element to ensure further learning.

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML</td>
<td>meta-learner</td>
</tr>
<tr>
<td>PD</td>
<td>pronouncing dictionary</td>
</tr>
<tr>
<td>ES</td>
<td>example selector</td>
</tr>
<tr>
<td>AC</td>
<td>accumulator</td>
</tr>
<tr>
<td>KB</td>
<td>knowledge base</td>
</tr>
<tr>
<td>TKB</td>
<td>temporary knowledge base</td>
</tr>
<tr>
<td>TR</td>
<td>translator</td>
</tr>
<tr>
<td>CH</td>
<td>ranking and checking procedure</td>
</tr>
<tr>
<td>LE-GS</td>
<td>learning English grapheme segmentation</td>
</tr>
<tr>
<td>LE-SR</td>
<td>learning English syllabification rules</td>
</tr>
<tr>
<td>LE-SS</td>
<td>learning English stress for syllables</td>
</tr>
<tr>
<td>LE-PG</td>
<td>learning English pronunciation for graphemes</td>
</tr>
</tbody>
</table>

Table 1: LEP Elements and their Purposes

The LU components are ordered according to the learning tasks. The LE-GS component is applied first because other learning components are based on the knowledge of graphemes. LE-SR learns how to
group graphemes into syllables. Then LE-SS places the stress on the syllables. Finally, LE-PG learns the pronunciation. Figure 2 shows the logical units of LEP. Interrelated components are surrounded by dashed rectangle boxes to form logical units. A logical unit is tied together by inner loops of operation. The diagram shows that the first logical unit selects examples for a particular learning task, and these examples are taken by the second logical unit (the learning unit LU) as input for a particular concept learning task. Rules that are produced by the learning unit are transferred to one of the evaluation elements, and eventually the active rules with high accuracies are placed into the knowledge base.

Figure 3 shows overall data flow and file transactions among the processes during the learning cycles. The symbols used to represent data flow directions, files, entities, and processes are indicated at the upper left corner of the diagram.

Information from NTC2 is filtered through ES where information useful for the current learning task is selected. This useful information is transformed into examples of the required format for a particular learning task. Learning examples are classified as positive and negative according to what is being learned. Then, these examples are taken by one of the learning elements to obtain sets of rules. These rules may be overlapping, overly generalized, or too specialized. With the help of the translator TR, the CH and AC components test each rule and obtain statistical information in local learning cycles. Rules that contribute to obtaining higher accuracy are accumulated in TDB and eventually transferred to the KB.

The software hierarchy chart in Figure 4 gives a conceptual view of how the LEP software is organized [19]. ML is the top level controlling program of CH, CLE, ES, and TR. There are also connections between modules within local loops but they are not shown in Figure 4 because a HIPO (Hierarchy Plus Input Process Output) chart is not designed to show sequences of execution but the partition among modules [11].

2.1 Meta Learner

The Meta Learner (ML) controls the other LEP learning processes (see Figure 4). It has four main functions: directing the learning processes, providing a suitable learning environment for every learning process, evaluating the learning results, and making decisions according to the feedback information. As mentioned, LEP has four major learning elements: LE-PG, LE-GS, LE-SR, and LE-SS. They are all under the control of ML. ML provides suitable learning examples for each learning element before starting a learning process. After every cycle of learning, ML calls the TR translator to translate randomly selected words and check the results. If the results meet the requirements of
ML, the learning process is continued. Otherwise, ML considers either changing some condition variables or increasing the number of condition variables for the input learning examples. According to the new set of condition variables, a complete new learning cycle starts. In this manner, ML runs the whole learning process automatically until no increase in learning accuracy.

2.2 The Environment

The learning environment includes an online pronouncing dictionary (NTC2) and an example selector (ES). The pronouncing dictionary is derived from the NETtalk Corpus, an on-line pronouncing dictionary used to train NETtalk [16]. Many inconsistencies were found during the sound symbol conversion process. These inconsistencies were corrected by hand and by a set of programs. The original IPA symbols and IPA representations [14] are used in NTC2.

Since LEP can be modified to learn other alphabetically written languages, it is important to use the IPA symbols consistently. If there is an IPA symbol for a particular sound, the IPA symbol is used instead of symbols defined by any other system. For example, the IPA symbol [j] is used to present the first sound in the word yellow. In the modified North American IPA system, the symbol [y] is used to represent this sound. But, in the IPA system, the symbol [y] is a “high front rounded” vowel sound [14], which is used in French and German to represent the <u> sound as in the words cru in French [2], and kürzes in German [18].

The example selector (ES) forms another part of the learning environment. Different learning components have different requirements for learning examples. For instance, learning to recognize graphemes needs examples of all kinds of graphemes, but learning to classify stresses needs examples of all types of stressed or unstressed syllables. It is ES’s responsibility to provide required positive and negative examples for each learning component. ES accepts commands from ML, including information about the kind of examples needed and the number of fields required in each example. We illustrate with an example of how ES selects examples for LE-PG.

Suppose, a dictionary contains a set of words W. For each grapheme g, ES selects the set of examples w_g where

\[ w_g = \{ w \in W \mid \text{grapheme } g \text{ appears in } w \} \]

All words in w_g contain at least one instance
of the target grapheme \( g \). In the NTC2 pronouncing dictionary, there are 132 graphemes and each grapheme represents between 1 and 14 different pronunciations. There are 366 different grapheme-pronunciation combinations and thus 366 sets of example words are needed to learn English pronunciation. ES performs the following steps to obtain examples for LEP-PG:

- obtain \( w_g \) by selecting all the words with the target grapheme from NTC2
- divide \( w_g \) into subsets according to their IPA symbols
- extract the information about the grapheme and put it in tabular form along with the IPA symbol
- treat those examples with same IPA symbol as the target IPA symbol as positive examples and the rest as negative examples.

### 2.3 The Learning Elements

The learning element of LEP consists of six components, which are shown in Figure 1. These components are classified into two groups according to what the component learns. The goal of LE-GS is to learn the structure of English graphemes in order to summarize the rules for constructing graphemes from words. LE-SR learns to syllabify English words and produces a set of syllabification rules. LE-SS learns to classify stresses for English words and produces a set of stress rules. The learning component LEP-PG learns to pronounce English graphemes by obtaining a set of pronunciation rules. The above four learning components all generate rules by learning. These rules can be seen as predicates that describe the concepts of graphemes, syllables, stresses, and pronunciations. Therefore, we call them the concept learning components of LEP. CH and AC have been implemented separately
for each learning component. Each learning component is described in a separate subsection.

2.4 The Knowledge Base

KB contains all rules learned by LEP. It has spaces for temporary databases as well as permanent databases. The temporary databases are produced by different LEP components. They are used, tested, and changed over and over until ML is satisfied with them. At that time, these databases become permanent databases which contain knowledge about graphemes, syllables, stresses, and pronunciations. This knowledge is embedded in four sets of rules: grapheme identification rules, syllabification rules, stress rules, and pronunciation rules.

2.5 The Performance Element

Since the primary purpose of LEP learning is to improve the performance of text-to-IPA translation, a flexible performance element is required. The performance element PE is designed not only to demonstrate the learning results but also to help improve learning.

PE includes two main parts: the translator (TR) and the converter (CV). As the performance element, TR is not only responsible for translating words to sound symbols after LEP has accomplished its learning tasks, but it is also involved in all learning stages. To assess learning progress and to provide feedback information, TR is applied in every cycle to translate the examples, using the rules that LEP has just learned.

CV is an IPA symbol converter. It interprets sound symbols between different systems. For example, the sound symbols used by LEP are different from those of used by NETtalk, and different speech synthesizers accept different kinds of sound symbols because there is no standard sound symbol system for every text-to-speech system to follow. CV can be seen as an IPA and sound symbol exchange table.

3 LE-PG Component

Using a machine-learning approach [12], based on the Version Space Algorithm (VSA) [13], LE-PG [20] was designed to obtain a set of complete pronunciation rules for the orthography of all (3,723) one-syllable words in NTC2.

The goal of LE-PG is to obtain a set of reliable pronunciation rules specified in terms of the IPA symbols. The structure of LE-PG is based on the Iterated Version Space Algorithm (IVSA) [4] [21] [22] [23].
The main idea of IVSA is to find regional hypotheses (RHs) of a learning concept, where a regional hypothesis is an ordered disjunctive description of a concept. First, IVSA uses the Generator algorithm to produce a set of candidate hypotheses from the set \( I' \) of input instances that are not yet covered. Then, the Assembler selects the most promising candidate hypotheses and orders them to form a single disjunctive hypothesis.

The Hypothesis Generator handles multiple values of the decision attribute. All instances are classified as positive and negative. For each decision value, we create a set \( X \) of appropriate instances by marking all instances with this decision value as positive and all others as negative. Each iteration of the inner loop generates multiple candidate RHs for a single decision value. Each RH is consistent with a series of instances. If an instance \( x_m \) is encountered that (according to VSA) would destroy the version space for \( X \), the Generator instead stores the existing specific hypothesis set (S) and the existing general hypothesis set (G) and then begins constructing a new RH, using the instances beginning with \( x_m \) as input. Thus, each RH is constructed from mutually consistent instances. Finding inconsistent instances of a concept provides a natural way of separating regions of a concept; these regions form the basis for a disjunctive concept hypothesis.

The Assembler first ranks the candidate RHs produced by the Generator. Ranking is done according to measure \( R = (|P_c| + |N_c|) / |I| \), where \( P_c \) is the set of positive instances that correctly match the hypothesis, \( N_c \) is the set of negative instances that correctly do not match the hypothesis, and \( I \) is the complete set of instances. \( R \) indicates the quality of an RH.

The Assembler then considers whether each candidate RH \( h \) should be added to the list \( H \) of accepted hypotheses, as follows. First \( h \) is added to \( H \) to form \( H' \). If the classification accuracy is higher with \( H' \) than with \( H \), then \( h \) is kept in the accepted list \( H \). If other hypotheses are already present in \( H \), then \( h \) is tested in all positions in the list, and the position resulting in the highest accuracy is selected. This process is repeated for each candidate RH. Given \( c \) candidate RHs and \( d \) accepted hypotheses, the Assembler performs \( O(c(c + d)) \) tests. Finally, the Assembler considers whether any accepted hypotheses could now be deleted because of other hypotheses that have been inserted in the list of accepted hypotheses.

In our experiments, the input was 3,724 one-syllable words from NTC2. These words include 17,723 unique grapheme examples, 98 different graphemes, and 46 different sound units. LE-PG was applied to 90% of the input instances (positive and negative) and tested on the unseen 10% instances (positive and negative) in ten separate runs (see Table 2). For the experimental run \( k \), the unseen instances were \( \{x_j \mid j \mod 10 = k\} \), i.e., \( x_1, x_{11}, x_{21}, \ldots \) for run 1. Correct IPA pronunciations were produced for 93.01% to 97.85% of the unseen words and 98.12% to 99.43% of the graphemes in the unseen words. For example, if the word *abacus* is translated as /æ/, b, ei, k, d, s/ instead of /æ/, b, d, s/, then 5 out of 6 graphemes are translated correctly, but 0 words are translated correctly.

The rule usage for the first run is shown in Figure 5. One of the rules learned on 90% of the instances was used 852 times when pronunciations were derived for these same 90% of the instances. Over 600 rules were used only once, and these rules mostly handled exceptions. The usage of the pronunciation rules learned by LE-PG follows both the “80-20 rule” [9], and Bradford-Zipf’s distribution [6].
<table>
<thead>
<tr>
<th>Run Number</th>
<th># of Words (Graphemes)</th>
<th>Accuracy</th>
<th># of Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Learned</td>
<td>Tested</td>
<td>Words</td>
</tr>
<tr>
<td>1</td>
<td>3,351 (12,598)</td>
<td>372 (1,403)</td>
<td>96.51%</td>
</tr>
<tr>
<td>2</td>
<td>3,351 (12,572)</td>
<td>372 (1,429)</td>
<td>95.16%</td>
</tr>
<tr>
<td>3</td>
<td>3,351 (12,589)</td>
<td>372 (1,412)</td>
<td>97.85%</td>
</tr>
<tr>
<td>4</td>
<td>3,351 (12,600)</td>
<td>372 (1,401)</td>
<td>95.16%</td>
</tr>
<tr>
<td>5</td>
<td>3,351 (12,614)</td>
<td>372 (1,387)</td>
<td>96.24%</td>
</tr>
<tr>
<td>6</td>
<td>3,351 (12,671)</td>
<td>372 (1,384)</td>
<td>93.01%</td>
</tr>
<tr>
<td>7</td>
<td>3,351 (12,603)</td>
<td>372 (1,398)</td>
<td>96.24%</td>
</tr>
<tr>
<td>8</td>
<td>3,350 (12,596)</td>
<td>373 (1,405)</td>
<td>94.64%</td>
</tr>
<tr>
<td>9</td>
<td>3,350 (12,631)</td>
<td>373 (1,370)</td>
<td>95.71%</td>
</tr>
<tr>
<td>10</td>
<td>3,350 (12,581)</td>
<td>373 (1,420)</td>
<td>95.98%</td>
</tr>
<tr>
<td>Average</td>
<td>3,351 (12,600)</td>
<td>372 (1,401)</td>
<td>95.65%</td>
</tr>
</tbody>
</table>

Table 2: Summary of Results

Figure 5: Rule Usage for LE-PG

4 Other Components

4.1 LE-GS

LE-GS learns rules for grapheme segmentation, i.e., identifying graphemes. The smallest written forms corresponding to the sound units are graphemes. English graphemes can be divided into three groups according to the number of letters they contain: 20% are single-letter, 63% are double-letter, and 17% are triple-letter graphemes. Most words formed by single-letter graphemes have the same number of graphemes as sounds. These words are tried first to obtain the basic general rules. The general rules are relatively easy to learn because the number of graphemes and sounds are the same. The rest of the words in NTC2 are harder to learn because multi-letter and silent graphemes are involved.

The input to LE-GS is a set of pairs, each including a word and the graphemes of the word. The output rules are based on a six position window, which begins with two positions reserved for the graphemes before the target letter combination and ends with positions reserved for letters after the target letter combination. Each grapheme rule has 3 groups of information: [Condition, Grapheme, Type]. The first group, Condition, includes six elements: next preceding letter (Npl), immediate preceding letter (Ipl), first target letter (Ftl), second target letter (Stl), immediate following letter (Ifl), and next following letter (Nfl). That is,

\[ \text{Condition} = [\text{Npl}, \text{Ipl}, \text{Ftl}, \text{Stl}, \text{Ifl}, \text{Nfl}] \]

The second group, Grapheme, is the aligned grapheme restricted by the Condition.
The last group, \textit{Type}, records whether the grapheme is a single-grapheme (1), a double-grapheme (2), or a triple-grapheme (3). Therefore, the format of a rule is:

\[ \text{rule}([N\text{pl}, I\text{pl}, F\text{fl}, S\text{ll, l\text{ll}, N\text{fl}]}, \text{Grapheme, Type}) \]

Examples:

1. \text{grapheme}([\_\_p, h, nil, \_], [ph], 2).
2. \text{grapheme}([\_\_nil, t, h, \_], [th], 2).
3. \text{grapheme}([\_\_b, \_\_\_\_\_\_], [b], 1).

These rules can be interpreted as:

\[ \begin{align*}
&\text{p.h} \rightarrow \text{ph} / \_\_\# \\
&t.h \rightarrow \text{th} / \# \_\_ \\
&b \rightarrow b / \_\_\_\_\
\end{align*} \]

4.2 LE-SR

LE-SR learns rules for dividing words into syllables [23]. The representation used in LE-SR is based on a common convention in linguistics whereby “C” represents a consonant grapheme (denoted as “-syllabic”) and “S” represents any vowel or syllabic consonant grapheme (denoted as “+syllabic”). Besides “C” and “S”, we found distinguishing consonant clusters from ordinary “-syllabic” graphemes useful for learning syllabification rules. Therefore, we let “CL” represent a consonant cluster and mark it as “-syllabic” as well. This representation is called “C-S-CL” and used in the first of three passes performed by LE-SR.

Ordinarily, all vowels are “+syllabic” except some of those represented by the graphemes \textlt{<e>} and \textlt{<y>}. This raises some problems. For example, in the word \textlt{eye}, the first \textlt{<e>} (together with \textlt{<y>}) is pronounced as /ai/, but the second \textlt{<e>} is silent. Because a silent sound cannot form a syllable alone, the last \textlt{<e>} belongs to the same syllable as \textlt{<ey>}. Another kind of problem is that the semi-vowel \textlt{<y>} is sometimes syllabic (in \textlt{pretty}) and sometimes non-syllabic (in \textlt{yes}).

To solve these two problems, LE-SR performs a second pass on the incorrectly translated patterns using \textlt{<e>} and \textlt{<y>} as separate symbols in addition to C, S and CL. That is, when LE-SR learns the most general rules, it uses the C-S-CL representation, and when it learns less general rules, it uses the C-S-CL-e-y representation. Finally, LE-SR performs a third pass to learn exception rules for the remaining cases. In this pass, a more specific representation is used, where each grapheme is represented separately.

In each pass of LE-SR, the first step is to identify all patterns between two syllabic graphemes in any input example. The second step is to determine where to separate the two syllables. Rules learned by LE-SR are organized in the format [\textit{Pattern, Cut, Num of syllabic}]. For example, \[[S, C, S, CL, S], [S, C, S], [CL, S]], 3] means whenever the pattern \[[S, C, S, CL, S] is encountered, LE-SR cut is between \[[S, C, S]\] and \[[CL, S]\] provided that the word has 3 syllabic graphemes. It is relatively easy for LE-SR to learn potential rules for dividing syllables since each observed pattern is a potential rule. However, it is very hard to determine which rules are good because one pattern can have several possible cuts. Consider the following pattern and its three possible cuts. The frequency is indicated in the third column of the examples:

\[ \begin{align*}
[[S, C, S], [[S, C, S]], 2]. & \quad 15. \\
[[S, C, S], [[S]], [C, S]], 2]. & \quad 1623. \\
[[S, C, S], [[S, C]], [S]], 2]. & \quad 45. \\
\end{align*} \]

The pattern is taken from example words containing two syllabic graphemes. To determine which cut should be chosen as a candidate rule, LE-SR determines the frequency for each cut. The frequency is indicated in the last column of the above ex-
ample. LE-SR chooses the cut that has the highest frequency. In our example, it is $[[S, C, S], [S, [C, S]], 2]$ with a frequency of 1,623. A detailed description and experimental results of LE-SR can be found in [23].

### 4.3 LE-SS

To learn stress rules for English words, we use the part of speech and the number of syllables. The stress on syllables for some English words varies depending on the part of speech. For example, as a noun, ‘record’ is stressed on the first syllable, while as a verb, it is stressed on the second syllable. The number of syllables influences the position of stresses. For example, the three-syllable word *horizon* has stress on the second syllable, but the four-syllable word *horizontal* has it on the third syllable.

The format of the rules is [Part, Nsyl, Pos, Stress], where ‘Part’ records the part of speech of the target word, ‘Nsyl’ is the number of syllables in the word, ‘Pos’ tells the position of the target syllable, and ‘Stress’ is the resulting stress for this particular kind of syllables.

### 5 Conclusions

In this paper, we described the LEP learning system with emphasis on the LE-PG component. We also provided 10 experimental results for LE-PG. A comparison of text-to-speech systems, including LEP and our use of C4.5, is shown in Table 3.

The results show that LEP-PG is an efficient concept learning system. The LEP technique is quite different from other research shown in Table 3. The main difference between the LEP technique and the traditional rule-based method is that the rules are obtained by the system automatically instead of being derived by hand. Unlike the neural network approach, the rules learned by LEP can be clearly displayed and exceptions are properly handled. There is no fixed number of levels in the learning space of LEP as with the Default Hierarchy approach. The number of levels of learning space of LEP can be as few as one and as many as hundreds, depending on the complexity of the concept learned.

The LEP learning system is designed to learn complicated English pronunciation, but it does not set the entire problem on the work-bench at once. It automatically
expands and contracts its learning space co-
incide with the various degrees of complex-
it y in English pronunciation. If a learning
space is too small, it cannot cover the tar-
get learning concept. On the other hand, if
the learning space is too large, learning will
be extremely slow.

LEP decomposes the English pronunciation
problem into small subsets of problems and then solves them one by one. As
this paper has described, LEP does not rely on morphophonemic, phonological and
prosodic rules. Instead, LEP automatically
produces pronunciation rules by learning
from positive and negative examples.

LEP produces a set of reliable pronunciation rules which obtains an average ac-
curacy of 95.65% for words and 98.79% for
graphemes when applied to unseen words.
The statistics on rule usage provide hard ev-
dence that English pronunciation is based
on rules.

Acknowledgements

This research was supported by the Natural
Sciences and Engineering Research Council
of Canada, the Institute for Robotics and
Intelligent Systems, and the University of
Regina.

References

editors. *From Text to Speech: The
MITalk System*. Cambridge University

English Dictionary*. Librairie Laroussu,

[3] H. Elovitz, R. Johnson, A. McHugh,
and J. Shore. Letter-to-sound rules for
automatic translation of English text
to phonetics. *IEEE Transactions on
Acoustics, Speech, and Signal Process-

iterated version space algorithm. In *Proc.
of Ninth Florida Artificial Intelli-
gence Research Symposium (FLAIRS-
96)*, pages 209–213, Daytona Beach,
Florida, May 1996.

and G. Papcun. A default hierarchy for
pronouncing English. *IEEE Transac-
tions on Pattern Analysis and Machine

cyclopedia of Library and Information
Science*. Marcel Dekker, New York,
1979.

[7] D.H. Klatt. The Klattalk text-to-
Acoustics Speech Signal Process-

[8] D.H. Klatt. How KLATTalk be-
came DECTalk: An academic’s expe-
rience in the business world. In *Offi-
cial Proceedings Speech Tech’87: Voice
Input/Output Applications Show and

Programming: Searching and Sorting.*

of Speech Production*. Pro.Ed, Austin,

Design of Business Information Sys-


