Abstract
This paper presents two letter-to-sound (LTS) methods in building a small-footprint multilingual text-to-speech (TTS) engine. For the languages where there exist a systematic relationship between a word format and its pronunciation, we employ a rule-based method. Otherwise, we use a training-based method. In the second method, we adopted optimal sequence to implement the process of letter-to-phoneme alignment, and use CART to train the decision tree and store the results in an efficient way. Despite their merits and disadvantages, the experimental results on six languages demonstrate these methods are effective to reach an acceptable LTS precision within a reasonable small size.

1. Introduction
Letter to sound module plays a very important role in a TTS system. Although many great efforts have been made on this problem[1,2,3,4], there still exits big research room when TTS becomes an eye-attractive feature in many devices. On one hand, in some embedded TTS applications, for instance, digit dialing and short message reading on a mobile phone, computational resource and memory occupation are the two key issues for successful application.

On the other hand, the globalization of industry requires the research of multilingual TTS. However, it is difficult for a person to create letter-to-sound module, because one is hard to be professional speaker of many languages.

For large TTS system, the orthodox method is to store a pronunciation lexicon with reasonable number of items. And only apply letter to sound module to those not listed in this lexicon. But for embedded system, it is not a smart way to use this method, because the lexicon will sharply increase the engine’s size. The improved method is for the most frequently used words, apply letter-to-sound, for the word with high usage frequency but can NOT get right prediction with letter to sound module, add them to an exception list.

The remaining parts of this paper are organized as follows. In section 2, we introduced rule-based letter-to-sound. In section 3, we presented training-based letter-to-sound. In section 4, employing English, French, Spanish, Portuguese, Italian and German as the example languages, we conducted the experiments. The last section is the conclusion.

2. Rule-based letter-to-sound
Typically, there are two possible solutions to build a LTS module: 1) manually collect rules, with sufficient experience and language knowledge. This method is commonly referred to rule-based method; 2) acquire rules by machine learning, which is called training-based method generally.

For a given language, if there exists a systematic relationship between a word format and its pronunciation, rule-based letter-to-sound can be efficient. During our research, we found Spanish, Italian, Portuguese and German are such kind of languages.

The workflow of rule-based letter-to-sound is as follows. Firstly, we invite experienced linguists to create a set of pronunciation rules for a language. The programmer translates the rules that are expressed by natural language into a set of machine-readable rules. At the same time, we collect a number of pronunciation items for testing the rules.

Secondly, the predicted items are compared with standard items to see what is the precision of the current rule set. If the precision is not high enough, we may modify the rule set and the codes. This interactive process is repeated until the precision reaches a predefined level within the acceptable code size.

2.1. General description
A general rule can be described as:

\[ [LS]+[CC]+[RS] \Rightarrow C_1C_2...C_i \rightarrow P_1P_2...P_j \] (1)

where \( CC \) means the current character, \( LS \) means left string, and \( RS \) means right string. \( C_iC_{i-1}...C_1 \) ( \( i \geq 1 \) and exists \( 0 \leq k \leq i \), \( C_1 = CC \)) are the characters belonging to the whole string, and \( P_1P_2...P_j \) ( \( j \geq 0 \)) means the phoneme string. The whole rule is read as: if the current character is \( CC \), and the left and right content are \( LS \) and \( RS \) respectively, then the characters \( C_iC_{i-1}...C_1 \) are pronounced as phoneme string \( P_1P_2...P_j \).

Note that \( LS \) and \( RS \) here could be empty and \( C_iC_{i-1}...C_1 \) could be a single character or a character string, while \( P_1P_2...P_j \) could be empty or one phoneme or more a phoneme string.

For example, in Spanish, we have:

\[ \# \text{sch} \Rightarrow \text{sch} \rightarrow g \] (2)

It is interpreted as: if current character is ‘s’, and the right following two characters are “ch”, then the sub word “sch” is pounced as “\( gr^{*} \)”(“gs” stands for ‘f’ in IPA), irrespective of the left content. Here, we use char ‘#’ for empty string.

Apparently, these rules are easy to be implemented through a “switch …case …” structure in C or C++, but one should pay special attention to the order of the rules. Generally speaking, you can implement the rules with long \( LS \) and \( RS \) first, and then the shorter ones.

2.2. General description
The tradeoff between the precision and module size is always a challenging problem in the LTS design, no matter what kind of LTS method you is used.
The following charts show the variation of close testing precision and letter-to-phoneme code size (not including syllable boundary and stress prediction) when the number of rules increases. Fig. 1-a and 1-b show the Spanish precision and code size when rules are added from initially 59 to finally 117. Fig. 2-a and 2-b show the Portuguese’s precision and code size when rules are added from initially 65 to finally 137. Fig. 3-a and 3-b show the Italian’s precision and code size when rules are added from initially 45 to finally 70.

From these languages, we know that the more rules are employed, the higher close testing precision we achieve. As a result, the code size is increased as well. The precision for any language tends to approach a constant asymptotically as code size increases.

3. Training-based letter-to-sound conversion

Training-based method shows its advantages where there is no language knowledge available, or the work to write pronunciation rule set systematically is too difficult. Many comparative experiments show that it can achieve higher accuracy than rule-based approach.

The typical processes of training method include: letter-to-phoneme alignment (or grapheme-to-phoneme alignment), model training and testing. Of course, for embedded applications, we always need to achieve tradeoff between model size and accuracy.

3.1. Letter-to-phoneme alignment

The conventional approach is to estimate the probability that one character \( c \) corresponds to one phoneme \( p \). The DTW (Dynamic Time Warping) algorithm can serve this purpose[5]. In this algorithm, if a character corresponds to 2 or more phonemes, we need to add a new phonetic symbol indicating this phoneme combination to the phoneme set; if a character doesn’t pronounce, an epsilon is assigned to it. So before performing formal alignment, we need a training lexicon where each character in each word is aligned with a phonetic symbol to compute the initial probability that a character pronounces a certain symbol. Then, we construct a matrix where row stands for character while column stands for phonetic symbol. Thus, we can arrive at an optimal path with maximum probability. This path can indicate the optimal letter-to-phoneme alignment. If we lack the training lexicon, we can preset some experienced values for the initial probability and conduct iteration to reach a reasonable result.

In practice, this approach has deficiency when a character corresponds to 2 or more phonemes. For example, in English,
x can correspond to two phonemes, ks (hereafter all English phoneme notations are in CMU phoneme set[6]) in the word, fax. If we add a symbol K to stand for ks, the pronunciation of the word cakes, k'ehks is converted to k'ehK. But neither k nor s can pronounce K. This case shows that adding new phonetic symbol can bring noise to training data in some cases.

Figure 4: Alignment procedure of fax and cakes
To solve this problem, we modify the alignment procedure to allow a character to correspond to two or more phonemes. Also a character can correspond to an epsilon. We set two variables i, j for character and phoneme, respectively. A character $c_i$ can correspond to a phoneme sequence $p_i \cdots p_j$ ($t \leq j$), if $t$ equals $j$, then $c_i$ corresponds to an epsilon. Alternatively, we need a two-dimensional array $Prob[i][j]$ to hold the maximum probability product from the starting point (0,0) to the node (i,j). Also we need another array $Path[i][j]$ to hold the optimal node sequence from the starting point (0,0) to the node (i,j). The induction process is presented as follows.

$Proh[i][j] = \max \{proh[i-1][k] \times Proh(p_i \cdots p_j | c_i) \}_{1 \leq t \leq j}$ \hspace{2cm} (3)

where $Proh(p_i \cdots p_j | c_i)$ is the probability that $c_i$ corresponds to $p_i \cdots p_j$. The optimal sequence can be described as:

$Path[i][j] = Path[i-1][k] \cup \{(i,j)\}$ \hspace{2cm} (4)

where $k = \arg \max_{i} \{Proh[i-1][k] \times Proh(p_i \cdots p_j | c_i) \}_{1 \leq t \leq j}$

3.2. Model training
The training procedure can adopt various learning methods such as decision tree, Bayesian network and neural network etc. to train a model. Recently, PBA (Pronunciation by Analogy) and SVM (Support Vector Machine) have received people’s attention [7]. They can reach the state-of-the-art precision for out-of-vocabulary words. However, PBA requires large storage of instance set; SVM is very time consuming. These shortcomings hamper their applicability to embedded device. In our experience, decision tree is very competitive in terms of precision, computational efficiency and space efficiency. Here we give some details of the training procedure for this method.

3.2.1. Train the decision tree
Exactly speaking, we employed a special kind of decision tree, CART (Classification and Regression Tree) to build up the learning machine[8,9]. We adopted a fixed-length sliding window (The left 4 characters and the right 3 characters as a pattern for the current character) as 7 questions. From the training lexicon, we can get a huge instance set. Each instance is organized as follows.

Current_character  Pattern  Phonetic_symbol

For each character $c_i$, we can get an instance set and compute its impurity $I_m$ based on the following equation.

$I_m = 1 - \sum_{n=1}^{N} \text{Pr}ob(p_s_n | c_i)^2$ \hspace{2cm} (5)

where $p_s_n$ is the $n$-th phoneme sequence corresponding to $c_i$.

Then we try the remaining questions and compute their respective impurity by employing equation (5). We choose the question with the maximum impurity reduction to split the current node. This recursive procedure will be terminated under the condition that no question is available or impurity cannot be decreased any more.

Figure 5: An example CART tree

3.2.2. Save the decision tree
Suppose the training procedure generates a CART tree illustrated in Figure 5, where each node (except the root node) contains three fields, including answer for the previous question, the successive question and the default class label (pronunciation), we can save it in the following way.

(1) Conduct preorder traversal
(2) Use the left delimiter and right delimiter to represent hierarchy.
(3) Omit the answer for root node and omit the question for leaf node.
(4) Ignore a child node which has the same class label with its parent node.

Thus, we can significantly reduce tree size.

3.3. Pronunciation prediction
To predict pronunciation for an input word, we adopt stack operation to access the CART tree. If a left delimiter is encountered, we perform push operation. Conversely, if a right delimiter is encountered, a pop operation is required. For each character in the word, we have a corresponding CART tree and contextual pattern. The prediction procedure can reach a proper node (until the prediction procedure cannot go ahead) for the given pattern via stack operation.
The class label of this node is treated as pronunciation of the character. After predicting all characters in the word, we can get word pronunciation.

4. Comparison of two methods

Though the strategies are different, the two methods to build letter-to-sound modules have the same essence: to “discover” the rule set, and to save and deploy it in an efficient way.

The merits of rule-based letter-to-sound are: 1) Simple for implementation. A “switch…case…” structural can solve the problem. For trainable method, the process to train the decision tree and prune it to reasonable size is a more complex task. 2) Clear logic. The rules can be clearly expressed, even people can read and understand the rules easily. But in a trainable method, the CART tree is usually stored only in a machine-readable format to save the storage space. 3) Fast, low runtime memory usage and low computational power. Actually, rule-based method is to use a sliding window with variable length to scan the whole string once, so you almost need no additional dynamic memory and very low computational power. However, for trainable method, letter-to-sound needs dynamic memory to hold the CART tree information. The amount of runtime memory is related to the size of CART tree. 4) Small training corpus. For rule-based letter-to-sound, one only needs a small testing lexicon to evaluate the precision. But for trainable method, one needs a large lexicon for generate the CART tree, as well as a corpus for evaluation. The size and accuracy of training lexicon directly affects the precision of letter-to-sound.

Rule-based letter-to-sound does have some disadvantages. First of all, it is language dependent. Languages that have obvious systematic relationship between word’s written format and its pronunciation are more suitable using rule-based methods, such as Spanish, Portuguese and Italian. It is reported that for English, if a rule-based letter-to-sound is applied, only a precision of 26% can be achieved[4]. Secondly, it needs strong language knowledge supporting. One needs rich language background to extract pronunciation rules, while in trainable methods the rules are generated implicitly from an aligned lexicon. Thirdly, The rule-based letter to sound has bad scalability. For example, if one application allows the TTS system to be 20MB, while another only allows 500KB. Rule-based method usually can only provide fixed size of letter-to-sound, while trainable method may generate different size of CART tree to provide different precision of phoneme prediction.

5. Experimental results

In our multilingual TTS engine, we employed rule-based method for Spanish, German, Italian and Portuguese, while take training-based method for English and French. For both methods, all the items are served as testing items.

Table 1 shows the rule-based experimental results on Spanish, Italian, Portuguese and German (German lexicon is a small name database which doesn’t contain compound words). Table 2 shows the training-based results on English and French.

![Image of Table 1](http://www.cstr.ed.ac.uk/projects/festival/manual/festival_17.html)

<table>
<thead>
<tr>
<th>Language</th>
<th>Precision</th>
<th>Size</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanish</td>
<td>93.72%</td>
<td>12.4KB</td>
<td>53,445</td>
</tr>
<tr>
<td>Italian</td>
<td>87.35%</td>
<td>11.1KB</td>
<td>21,160</td>
</tr>
<tr>
<td>Portuguese</td>
<td>83.78%</td>
<td>15.3KB</td>
<td>25,962</td>
</tr>
<tr>
<td>German</td>
<td>92.00%</td>
<td>16.2KB</td>
<td>8,646</td>
</tr>
</tbody>
</table>

Table 2: Training-based experimental results

<table>
<thead>
<tr>
<th>Language</th>
<th>Precision</th>
<th>Size</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>85.0%</td>
<td>180KB</td>
<td>104,000</td>
</tr>
<tr>
<td>French</td>
<td>90.3%</td>
<td>58KB</td>
<td>320,000</td>
</tr>
</tbody>
</table>

6. Conclusions

This paper presents two methods for building letter-to-sound in small-footprint multilingual TTS engines. We employ rule-based method for Spanish, German, Italian and Portuguese, while taking training-based method for English and French. The experimental results showed that these methods are suitable for the languages we have processed. However, we feel there may be room to improve the precision while keeping or even cutting the current code size.

7. References


