Exact Generalization of Finite-State Transductions: Application to Grapheme-to-Phoneme Transcription

Yves Schabes, Emmanuel Roche

TR95-08   December 1995

Abstract

We present two methods for building a finite-state transducer which generalizes a finite-state transduction to any word on a given alphabet. The methods are exact in the sense that the inferred transducer coincides on the inputs for which the initial function is defined. We apply the methods to the problem of grapheme-to-phoneme transcription. In this case, the initial function is given in the form of an aligned letters–phonemes dictionary. The generalized function gives a phonetic transcription for words not in the dictionary based on its similarity to other words. In addition, for this problem, the generalization could be represented more compactly than the initial dictionary.

Revisions history.

1. Version 1.0, 95/03/16

1 Introduction

Finite-state devices have important applications to many areas of computer science, including pattern matching, databases and compiler technology. Although their linguistic adequacy to natural language processing has been questioned in the past, there has recently been a dramatic renewal of interest in the application of finite-state devices to several aspects of natural language processing. This renewal of interest is due to the speed and the compactness of finite-state representations. This efficiency is explained by two properties: finite-state devices can be made deterministic, and they can be turned into a minimal form.

Our work relies on two central notions: the notion of a finite-state transducer and the notion of a subsequential transducer. Informally speaking, a finite-state transducer is a finite-state automaton whose transitions are labeled by pairs of symbols. The first symbol is an input token and the second is an output token. Applying a finite-state transducer to an input string consists of following a path according to the input tokens while storing the output tokens, the result being the sequence of output tokens stored. Finite-state transducers can be composed, intersected, merged with the union operation, sometimes made deterministic and minimized[1]. Basically, one can manipulate finite-state transducers as easily as finite-state automata.[2]

There is a natural correspondence between annotated corpora and functions: a corpus can be seen as a collection of points and their images by a function that maps the raw input to the annotated output. Although finite-state transducers seem therefore appropriate for data-driven methods, finite-state transducers have seldom been used as a framework for automatic data-driven methods.

We present two methods for building a deterministic finite-state transducer which generalizes a deterministic function to any word on a given alphabet. In addition to their ability to generalize from examples, the methods are exact in the sense that the inferred transducer coincides on the inputs for which the initial function is defined.

The approach is general but for explanatory purposes in this abstract we describe it in the application context of grapheme-to-phoneme transcription. Using a 480,000-word aligned phonetic dictionary (Laporte, 1988) as training corpus[3], we show how a deterministic finite-state transducer that transcribes words to their phonetic transcriptions can be constructed. This transducer does not merely represent the dictionary since it is capable of handling unknown words accurately.

Numerous frameworks have been proposed for this problem, among which: purely rule-based systems (Allen et al., 1987; Kaplan and Kay, 1994), information theoretic systems such as decision trees (Lucassen, 1983; Lucassen and Mercer, 1984), hybrid systems (Meng et al., 1994; Huang et al., 1994), table lookup models (Bosch and Daelemans, 1993), dictionary-based (Coker et al., 1990; Laporte, 1993)[4] and neural networks (Sejnowski and Rosenberg, 1987).[5] However, to our knowledge, our method combines features that no other system embodies simultaneously: the method is exact in the sense that it covers precisely the full (480,000 words) phonetic dictionary given as training data; it generalizes since it performs well on new words not found in the training set; it is compact since the space required to store the function is smaller that

[1]See for example, Berstel (1979), Mohri (1994) and Roche and Schabes (1994).

[2]However, whereas every finite-state automaton is equivalent to some deterministic finite-state automaton, there are finite-state transducers that are not equivalent to any deterministic finite-state transducer. Transductions that can be computed by some deterministic finite-state transducer are called *subsequential functions*.

[3]In this paper, we only consider words with exactly one phonetic transcription.

[4]Laporte (1993) also gives a proof that the set of phonetic rules used in his system is equivalent to a finite-state transducer.

[5]We refer the reader to Klatt (1987) for an extensive survey of text-to-speech systems.

then space needed to store the phonetic dictionary; it uses a single framework for representing rules and the list (dictionary) of exceptions; it is bidirectional (letter-to-sound/sound-to-letter) since finite-state transducers are by definition bi-directional and it is very fast (12,000 words/s) since the letter-to-sound function is represented as a deterministic finite-state transducer.

We show how the first method can also be used for compression and evaluate it on unknown words. We then describe a second method for inferring an exact generalization in the form of a deterministic finite-state transducer. This method is closely related to approaches that transcribe each letter based on surrounding letters (such as for example, decision trees (Lucassen, 1983), neural networks (Sejnowski and Rosenberg, 1987) and table lookup models (Daelemans and Bosch, 1993)). The method is more accurate than the first one but is not useful for compressing the initial dictionary.

In the final paper, we will also report experiments on English data.

2 Preliminary Concepts

We illustrate the methods by applying them to the problem of grapheme-to-phoneme transcription. For experimental purposes, we use an aligned dictionary of 480,000-word French words (Laporte, 1988). The dictionary consists of a list of words where each character is associated with a one-character phoneme code (possibly representing an empty phoneme). In this paper, we only consider words with exactly one phonetic transcription.

For example, the French word "`artichaut`" ("artichoke" in English) is transcribed as "`artiS--o-`" where "–" stands for the empty sound; we write `artichaut/artiS--o-`.

The methods generalize the data found in the dictionary using all aligned substrings of different length of all dictionary entries. Substrings of n characters (*n-grams*) aligned with n phonemes are called *n-phons*. Given an aligned dictionary, the set of *n-phons* are trivially computed.

For example, all *4-phons* for the above example are `arti/arti`, `rtic/rtiS`, `tich/tiS-`, `icha/iS--`, `chau/S--o` and `haut/--o-`.

For a given length n, the methods build a dictionary of *n-phons* that correspond to the most likely transcription of each *n-gram*. For example, if the *3-phon* `cha/S--` occurs five times in the dictionary and the *3-pho* `cha/S-a` occurs three times in the dictionary, the most likely *3-phon* for the *n-gram* `cha` is `cha/S--` and only this one will be stored in the *3-phon* dictionary.

The construction and the use of these dictionaries differ for both method and will be described in the following two sections.

3 First Method

The method is better illustrated by an example. Consider the following sample dictionary obtained by merging *1-phon*, *2-phon* and *3-phon* dictionaries:[6]

[6]In practice it is useful to add begin and end markers to each word. For sake of simplicity those markers are ignored in this paper.

```
a/a
r/r
t/t
i/i
c/k
h/-
u/-
ch/S-
au/-o
aut/-o-
```

Using the above dictionary, the word `artichaut` is transcribed left to right as follows.

$$\frac{\texttt{a}}{\texttt{a}}\quad\frac{\texttt{r}}{\texttt{r}}\quad\frac{\texttt{t}}{\texttt{t}}\quad\frac{\texttt{i}}{\texttt{i}}\quad\frac{\texttt{ch}}{\texttt{S-}}\quad\frac{\texttt{aut}}{\texttt{-o-}}$$

Looking at the first letter (`artichaut` remains to be transcribed), the longest *n-phon* defined in the dictionary that matches what follows is `a/a`. `a` is emitted and we move to the second letter (`r`) and so on until the fifth letter. When we reach the fifth letter (`chaut` remains), we have emitted (`arti`). At that point, two *n-phons* match, `c/k` and `ch/S-`. The longest one, `ch/S-`, is chosen and we move to the seventh letter (`aut` remains) while emitting `S-`. At that point, `au/-o` and `aut/-o-` match. The longest one is chosen (`aut/-o-`) and the word `artichaut` is correctly transcribed as `artiS--o-`:

Note that in the above *n-phon* dictionary, the *5-phon* `chaut/S--o-` does not need to be stored since it can be obtained from this dictionary (with `ch/S-` and `aut/-o-`) with the same procedure.

More precisely, the method builds a series of *n-phon* dictionaries. The *1-phon* dictionary is first built.[7] Dictionaries of increasing lengths of *n-phons* are inductively built. Assuming that all dictionaries up to a given length $n-1$ have been built, we build the dictionary for length n by removing from the set of most likely transcriptions of length n the ones that can correctly be derived from the dictionaries of length up to $n-1$.

In our example, the *two-phon* `ar/ar` is not stored since it can be derived from the *uni-phons* `a/a` and `r/r`.

We could carry this operation up to the length of the longest word in the original dictionary, or stop at a shorter length k and construct an exception dictionary of entries that cannot be correctly derived from the *k-phon* dictionaries. Note that the exception dictionary only contains words of length strictly greater than n.

So far, we have described the method in terms of dictionaries (*n-phon* and exception dictionaries). The method however operates on finite-state transducers representing these dictionaries. Each *n-phon* dictionary can be trivially represented as a finite-state transducer where each arc is labeled with a pair of letter and phoneme. Similarly the exception dictionary is represented as a finite-state transducer.

We build inductively a single finite state transducer F_k that computes the phonetic transcription of a word. The transcriptions according to the *one-phon* dictionary is obviously a deterministic finite-state transducer. Inductively, assume that we have built the machine operating with the $1-$ to *n-phon* dictionaries. Suppose also that the *n+1-phon* dictionary is represented as deterministic finite-state transducer, then the algorithm described in Appendix A

[7] As previously said, this dictionary contains the most likely transcription of each character.

n	# exceptions	# states in F_n	$\|F_n\|/\|F_0\|$ % % compression
0	479,300	48,244	100
1	457,657	47,842	99
2	368,044	46,829	97
3	276,249	43,210	89
4	182,311	40,461	83
5	135,882	45,666	94

Figure 1: First method: different levels of *n-phons* applied to the whole dictionary.

largest *n-phon*	% letters	% words
1	71	4.4
2	85	23
3	91	43
4	94.5	61
5	96	69.5
6	96.7	74.2

Figure 2: First method: performance on unknown words

computes the combined deterministic transducer. This operation is repeated until the *n-phons* of length k are included. Finally, the exception dictionary is combined similarly (at the last step)[8] to form F_k.

By construction, the final deterministic finite-state transducer outputs the exact same transcription for each word in the initial phonetic dictionary. Moreover, it generalizes the function to any string. The constructed *n-phon* dictionaries can be seen as transcription rules inferred from the dictionary; and the exception dictionary as exceptions to those rules.

3.1 Experiments

We first ran experiments on the entire dictionary. Figure 1 shows the coverage and compression obtained by the transducer F_n constructed for increasing values of n (as described in the previous Section). The second column shows the number of words in the exception dictionary. The third column shows the number of states of the final deterministic transducer F_n (it includes the exception dictionary). Therefore, for any $n \geq 0$ and for any word w in the original dictionary, $F_n(w)$ is the original phonetic transcription. The fourth column shows the ratio of number of states in F_n to the number of states of F_0 (which is the original transducer representing the dictionary). This percentage represents the compression ratio. This compression ratio is based on the finite-state representation which already compresses by a factor of 10 compared to the ASCII representation. An optimal compression ratio (83%) is obtained for $n = 4$.

In order to evaluate the generalization capabilities of this method, we also ran experiments by randomly splitting the dictionary into two parts, 90% of which was used for training purposes and 10% for testing. Figure 2 shows the performance obtained by the transducer constructed for increasing values of n. The second column shows the percentage of letters correctly transcribed and the third column the percentage of words correctly transcribed.

The experiments show that after a certain length (4), there is a tradeoff between the compression and generalization capabilities.

[8]The exception dictionary contains *n-phon* of greater than k.

We can further improve the generalization capabilities by using an alternative method described in the next section.

4 An Alternative Method: Window Sliding

The first method transcribes *n-grams* of characters at a time from left to right while always trying to transcribe as many characters at a time. When an *n-gram* has been transcribed, the characters in this *n-gram* are no longer considered when transcribing the remaining characters. The second alleviates this apparent drawback and is capable of considering any character when making a decision.

The method is best understood by an example. Consider the following *n-phons* dictionary:

```
fila/fila
lament/lamã--
ment/m---
cla/kla
```

The first method transcribes the word `filament` incorrectly to `filam---`:

```
fila    ment
fila    m---
```

It considers the first letter and matches `fila/fila` which is the longest *n-phon* in the dictionary for that letter; it then moves to the fifth letter (`ment` remains) and matches `ment/m---` and moves to the end. This method seems inappropriate in this case since it did not use the longer context `lament/lamã`.

The second method transcribes the word `filament` correctly to `filamã--` as follows.

```
filament
f···
·i··
  l·····
  ·a····
  ··m···
  ···ã··
  ····-·
  ·····-
```

The first letter `f` is considered and the longest match around this letter is used to transcribe this letter. The longest match is `fila/fila` and `f` is transcribed as `f`. Then, the second letter `i` is transcribed as `i` since `fila/fila` is the longest match that can be slided and aligned on `i`. Then, the third letter `l` is consider. The longest match that can be slided on `l` is `lament/lamã--`. Similarly, `a` is transcribed to `a` according to `lament/lamã--`; `m` is transcribed to `m` according to `lament/lamã--`; `e` is transcribed to `ã` according to `lament/lamã--`; `n` is transcribed to `-` according to `lament/lamã--` and `t` is transcribed to `-` according to `lament/lamã--`.

However, using this dictionary, both methods will correctly transcribe `clament` to `klam---`.

More precisely, the second method transcribes an input word one character at a time. For decreasing window lengths, the method first aligns the *n-phon* window starting at the current character and slides the window until a match is found. When the match is found, the current character is transcribed according to the *n-phon* and the method is reiterated on

	First Method				Window Sliding			
n	# exceptions	$\|F_n\|/\|F_0\|$ %	% letters	% word	# exceptions	$\|F_n\|/\|F_0\|$ %	% letters	% word
0	479,300	100	0	0	479,300	100	0	0
1	457,657	96	71	4	457,657	96	71	4
2	368,044	79	85	23	369,135	78	85	23
3	276,249	59	91	43	241,053	50	93	50
4	182,311	39	95	61	165,452	35	95	64
5	135,882	32	96	70	116,752	27	96	72
6			97	74	75,930	21	97	79
7					54,279	19	98	82
8					45,645	18	98	83
9					41,973	19	98	84

Figure 3: Second method. Comparison with Window Sliding.

another untranscribed character. This transcription can be done in any order, not necessarily from left to right.

This method bares resemblance to previous approaches since it looks at windows of characters in order to transcribe the input.

Similarly to the first approach, we build the *n-phon* inductively on length n. The *1-phon* dictionary is identical to the one of the first method. The *n+1-phon* dictionary is obtained by removing any *n+1-phon* correctly encoded (according to the second method) by the dictionary of $1-$ to *n-phons*.

Figure 3 compares the two approaches. In this Figure, $|F_k|$ is the number of characters in the ASCII dictionary of *1-* to *k-phons* plus the number of characters in the exception dictionary (in ASCII form). The Window Sliding method has a better generalization accuracy. This results in a simple compression method on the ASCII file. However, since this compression ratio is given with respect to the original ASCII file, it cannot be compared with the compression ratio of Figure 2 which was given with respect to the finite-state representation of the original dictionary.

Due to the lack of space, we do not include more details on this approach. The final paper will include more details on this method. In particular, we will show how this method can be turned into a deterministic finite-state transducer.

5 Conclusion

There is a natural correspondence between annotated corpora and functions: a corpus can be seen as a collection of points and their images of a function that maps the raw input to the annotated output. We have shown that finite-state transducers are very well suited for data-driven methods.

As a case study, we experimented with grapheme-to-phoneme transcription using a 480,000 word phonetic dictionary of French. We have presented two methods for building a deterministic finite-state transducer which is capable to transcribe any word even of it is not in the original dictionary. In addition to their ability to generalize from examples, the methods are exact in the sense that the inferred transducers coincide on the words defined in the dictionary. Since the resulting transducers can be smaller than the original function, the methods can also be used for dictionary compression.

The two methods differ on their generalization accuracy and their compression capabilities.

The methods are general and we believe that they are useful for a wide range of data-oriented applications.

References

Jonathan Allen, Sharon Hunnicutt, and Dennis Klatt. 1987. *From text to speech : the MITalk system.* Cambridge University Press.

Jean Berstel. 1979. *Transductions and Context-Free Languages.* Teubner, Stuttgart.

Antal van den Bosch and Walter Daelemans. 1993. Data-oriented methods for grapheme-to-phoneme conversion. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics (EACL'93)*, pages 45–53.

Cecil H. Coker, Kenneth W. Church, and Mark Y. Liberman. 1990. Morphology and rhyming: Two powerful alternatives to letter-to-sound rules for speech synthesis. In *Conference on Speech Synthesis.* European Speech Communication Association.

Walter Daelemans and Antal van den Bosch. 1993. TabTalk: reusability in data-oriented graphene-to-phoneme conversion. In *Eurospeech*, Berlin.

Caroline B. Huang, Mark A. Son-Belt, and David M. Baggett. 1994. Generation of pronunciation from orthographies using transformation-based error-driven learning. In *ICSLP94*, Yokohama, Japan, September.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Dennis H. Klatt. 1987. Review of text-to-speech conversion for English. *Journal of the Acoustical Society of America*, 82(3):737–792, September.

Eric Laporte. 1988. *Méthodes Algorithmiques et Lexicales de Phonétisation de Textes.* Ph.D. thesis, Université Paris 7.

Eric Laporte. 1993. Phonétique et transducteurs. Technical report, Université Paris 7, June.

John M. Lucassen and Robert L. Mercer. 1984. Discovering phonemic base forms automatically: an information theoretic approach. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 42.5.1–42.5.4.

John M. Lucassen. 1983. Discovering phonemic base forms automatically: an information theoretic approach. Technical Report RC 9833 (#43527), IBM T.J. Watson Research Center, February.

Helen M. Meng, Stephanie Seneff, and Victor Zue. 1994. Phonological parsing for bi-directional letter-to-sound/sound-to-letter generation. In *ARPA Human Language Technology Workshop*, Princeton, USA.

Mehryar Mohri. 1994. Compact representation by finite-state transducers. In 22^{nd} *Meeting of the Association for Computational Linguistics (ACL'94)*, pages 204–209.

Emmanuel Roche and Yves Schabes. 1994. Deterministic part-of-speech tagging with finite-state transducers. Technical Report TR-94-07, Mitsubishi Electric Research Laboratories, Cambridge, USA, June.

T.J. Sejnowski and C. R. Rosenberg. 1987. Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168.

A Combine Operation

In the following, a finite-state transducer T is formally defined as a 5-tuple (Σ, Q, i, F, E) where: Σ is a finite alphabet; Q is the set of states or vertices; $i \in Q$ is the initial state; $F \subseteq Q$ is the set of final states; $E \subseteq Q \times \Sigma \cup \{\epsilon\} \times \Sigma^* \times Q$ is the set of edges or transitions.

This leads to the definition of *deterministic* transducers called *subsequential* transducers: a subsequential transducer T is a 6-tuple $(\Sigma, Q, i, F, E, \rho)$ where: Σ, Q, i, F, E are defined as above, but E is such that $\forall q \in Q, \forall a \in \Sigma, |\{(q, a, b, q') \in E\}| \leq 1$; and the final emission function ρ maps F on Σ^*, one writes $\rho(q) = w$.

In addition, it is useful to define the *state transition function* d by $d(q, a) = q'$ s.t. $\exists (q, a, b, q') \in E$; and the *emission function* δ by $\delta(q, a, q') = b$ if $(q, a, b, q') \in E$.

For $w_1, w_2 \in \Sigma^*$, $w_1 \wedge w_2$ denotes the longest common prefix of w_1 and w_2.

The following algorithm combines two transducers $T_1 = (\Sigma, Q_1, i_1, F_1, E_1, \rho_1)$ and $T_2 = (\Sigma, Q_2, i_2, F_2, E_2, \rho_2)$ to form a third transducer $T = (\Sigma, Q, i, F, E, \rho)$ according to the method described in Section 3.

```
1. COMBINE_TRANSDUCERS($T_1 = (\Sigma, Q_1, i_1, F_1, E_1, \rho_1)$,$T_2 = (\Sigma, Q_2, i_2, F_2, E_2, \rho_2)$)
2. $q = 0; n = 1; i = 0; C[0] = ((0, \epsilon), (0, \epsilon)); F = \{0\}; \rho(0) = \epsilon; E = \emptyset;$
3.     do {
4.         $((x_1, u_1), (x_2, u_2)) = C[q];$
5.         if $(x_1 \in F_1)$
6.             $F = F \cup \{q\}; \rho(q) = u_1 \cdot \rho_1(x_1);$
7.         for each $a$ s.t. $d_1(x_1, a) \neq \emptyset$
8.             CASE 1 : $x_2 \neq$ OUT
9.                 CASE 1.1 : $d_2(x_2, a) \neq \emptyset$
10.                    $y_1 = d_1(x_1, a); y_2 = d_2(x_2, a);$
11.                    CASE 1.1.1 : $y_2 \notin F_2$
12.                        $\mu_1 = \delta_1(x_1, a, y_1); \mu_2 = \delta_2(x_2, a, y_2);$
13.                        $b = u_1 \cdot \mu_1 \wedge u_2 \cdot \mu_2;$
14.                        $S' = ((y_1, b^{-1} \cdot u_1 \cdot \mu_1), (y_2, b^{-1} \cdot u_2 \cdot \mu_2))$
15.                        if $\exists r \in [0, n-1]$ s.t. $C[r] == S'$
16.                            $e = r;$
17.                        else
18.                            $C[e = n++] = S';$
19.                        $E = E \cup \{(q, a, b, e)\};$
20.                    CASE 1.1.2 : $y_2 \in F_2$
21.                        $b = u_2 \cdot \delta_2(x_2, a, y_2) \cdot \rho_2(y_2);$
22.                        $S' = ((0, \epsilon), (0, \epsilon));$
23.                        if $\exists r \in [0, n-1]$ s.t. $C[r] == S'$
24.                            $e = r;$
25.                        else
26.                            $C[e = n++] = S';$
27.                        $E = E \cup \{(q, a, b, e)\};$
28.                CASE 1.2 : $d_2(x_2, a) = \emptyset$
29.                        Call AUX;
30.            CASE 2 : $x_2 =$ OUT
31.                Call AUX;
32.        $q++;$
33.    } while $(q < n);$
34. Function AUX
35.    $y_1 = d_1(x_1, a); b = u_1 \cdot \delta_1(x_1, a, y_1);$
36.    if $y_1 = 0$ then $y_2 = 0$ else $y_2 =$ OUT;
37.    $S' = ((y_1, \epsilon), (y_2, \epsilon));$
38.    if $\exists r \in [0, n-1]$ s.t.$C[r] == S'$
39.        $e = r;$
40.    else
41.        $C[e = n++] = S';$
41.    $E = E \cup \{(q, a, b, e)\};$
```