

Building Non-Native Pronunciation Lexicon for English Using a Rule Based Approach

Rohit Kumar
LTRC, IIIT Hyderabad,
Gachibowli,
Hyderabad 500 019
Andhra Pradesh, INDIA
rohit@iiit.net

Amit Kataria **Sanjeev Sofat**
Dept of Computer Science & Engineering,
Electronics Block, Punjab Engineering College,
Sector 12, Chandigarh 160 012
INDIA
amit_pec@email.com, chestasofat@yahoo.com

Abstract

Lexicon building is a time consuming and primarily manual process. Pronunciation Lexicons change depending upon the accent being used in different regions. We observe that the changes in pronunciations follow certain rules and the rules can be exploited to reuse the existing pronunciation lexicons to build non-native pronunciation lexicon. A rule based approach to build non-native pronunciation lexicon for English has been proposed through this paper. Also an example based approach to build up the rules is described.

1. Introduction

Pronunciation Lexicon is invariably used in the text processing modules of Text to Speech (TTS) Systems of languages in which the pronunciation of words do not have a simple relationship to the sounds of the language. Pronunciation Lexicon (or simply Lexicon) is used in two different ways in the text processing modules of TTS Systems – for pronunciation lookup for words in the Lexicon and for building models to predict the pronunciation of words not available in the Lexicon like proper nouns etc. Also Lexicons are used in building Language Models, which are required for Automatic Speech Recognition (ASR) Systems and several other Language Processing systems.

Pronunciation of language varies from region to region. Especially in regions where the language in consideration is a foreign language, its pronunciation is largely influenced by the pronunciation of the native language of the region. Typically the speakers try to replace the constraints of the foreign language by the constraints of their native language [Witt and Young, 1999]. For example, Indian speakers try to replace the stress timed pronunciation constraints of English by the syllable-timed constraints of most of Indian Languages giving rise to varying accents. But it is observable that this influence of native accent on the foreign language follows a set of rules that map the sounds of the foreign language to sounds of the native accent. Though the mapping is not straightforward it is derivable by observation.

Lexicon building is a time consuming, laborious and costly process. So it is desirable to reuse the information available in Native Pronunciation Lexicons to get the non-native pronunciation of the language.

Through this paper, a Rule-based approach to build non-native pronunciation lexicons from existing native pronunciation lexicons is proposed. The rules use the phonetic properties of the native phonemes to generalize the contextual effects under the influence of which a source phoneme is mapped to a phoneme in the target set of phonemes. Though the process of rule building is basically manual, we also describe a method to automatically build the rules through an example based approach to ease the process of rule building.

It should be mentioned that there are earlier works in the direction of non-native lexicon building. In [Downey and Wiseman, 1998; Cremelic and Martens, 1997] different rule based approaches are proposed that generate pronunciation variations from base pronunciations. In [Amadal et al., 2000] a data driven approach to automatically come up with pronunciation variations is described. [Fitt and Isard, 1999] uses a phoneme recognizer to generate English pronunciation of German words and to automatically train decision trees for predicting English accented variant of German. [Goronzy et al., 2001] describes the approach to use an accent-independent keyword lexicon to synthesize speakers of different regional accents. We focus on developing a rule based approach to come up with Non-Native English pronunciation and attempt to make the process of rule building flexible by proposing an example based approach.

The paper is organized as follows: Section 2 introduces our approach to use rules for generating the pronunciation variations and describes the rule format. Section 3 & 4 detail the methods to apply the rules on native pronunciation. In Section 5 the manual and example based approaches for rule building are described.

2. Overview of the Rule based Approach

A three-step process is followed in obtaining the non-native pronunciation for each word. First the word along with its part of speech is looked up in the native lexicon to obtain its native pronunciation as a vector of phonemes. In the second step the orthographic representation of the word is aligned with the phonological representation obtained in step 1. In English, the orthographic representation is made up of sequence of graphemes in the word spelling. The phonological representation is the vector of phonemes in the word's pronunciation. Alignment involves introduction of blank phonemes and blank grapheme in the pronunciation and spelling respectively. Details of the heuristics algorithm to align graphemes to phonemes are discussed in section 3. Finally the rules are applied onto the phoneme vector obtained after step 2 to get the corresponding non-native pronunciation as a vector of phonemes from the target set of phonemes. The criteria for firing a rule and other issues related to rules are discussed in section 4. Following is an example of the three-step process applied on the word *lieutenant*.

Step 1: Word Lookup: *lieutenant* (noun)

Pronunciation: l e f t e n @ n t

Step 2: Grapheme - Phoneme Alignment

l	i	e	u	-	t	e	n	a	n	t
l	%	e	%	f	t	e	n	@	n	t

% : Inserted Blank Phoneme

- : Inserted Blank Grapheme

Figure 1. Grapheme-Phoneme Alignment of *lieutenant*

Step 3: Rule Application

l	%	e	%	f	t	e	n	@	n	t
l	%	e	%	f	tz	e	n	e	n	tz

Figure 2. Source to Target Phoneme Conversion for *lieutenant*

It may be noted that the conversion from native phonemes to target phonemes is not a one to one function but rather a many to many function. For example, the schwa (@) may convert to a, e, ai, ez depending on its context.

The rules used in step 3 may be built up manually for the conversions which are observable straight forward and an example based rule building method can be used for further pruning the rules as discussed in section 5.

2.1. Format of Rules

Primarily the rules convert phonemes from native (source) phoneme set to non-native (target) phoneme set. In this section the format of the rules we have used and their features are described. In our effort we have used UK English Phoneme set as the source phoneme set as it is the phoneme set used by native Festvox Oxford's Advanced Learner's Dictionary being used by us. The source phoneme set with several relevant properties is given in Appendix A. The target phoneme set is based on sound in Indian accented English and we have used the symbols from Z-Notation [Kishore et al., 2002].

The rules consist of five fields: Left Context (LC), Source Phoneme (PH), Right Context (RC), Grapheme (GR) and Target Phoneme (OPH). LC, PH and RC are expressions consisting of phonemes or wildcards. The phonemes in the expression are separated by OR (|) operator. GR is another expression consisting of string of English alphabets and wildcards. As in LC, PH and RC, in GR also the OR (|) operator is used as separator. The OPH field is a simple string indicating the output to be generated when a particular rule fires. Each field is separated from the other by semicolons.

Each of the phoneme property mentioned in Table 1 has a 32 bit integer value corresponding to it. The integer value is equal to 2^N where N is the bit number associated with that property. In fact each one of low order 29 bits of the 32 bit integer represents a phoneme property. So the 6th bit (N=5) represents a schwa (SW) and is associated with the value 32 ($=2^5$). Also each phoneme in the source phoneme set described in Appendix A has a 32-bit integer value corresponding to it. This value is the sum of values associated with the properties of the phoneme. For example the phoneme p is a stopped labial consonant. So the

value corresponding with it is the sum of values corresponding to CS, SC and LB i.e. $20 + 213 + 219 = 532481$. In LC, PH, RC and GR expressions we use asterisk (*) as a wildcard. In LC, PH and RC expressions its value is taken to be equal to $2^{32} - 1$. The overall value of the LC, PH and RC expressions can be computed as the bitwise OR of the value associated with all the phonemes in these expressions.

So it turns out that in a rule comprises of three 32 bit integer values, one string expression and one output string. For example the rule:

[*]; [%]; [*]; {r}|{re}; [r];

implies that whenever a blank phone appears with a “r” or “re” as the grapheme corresponding to it, then in Indian Accent English a “r” sound is present. The LC, PH and RC are equal to the three integer values corresponding to the wildcard * and the blank phoneme %. The use of integer values for the LC, PH and RC fields makes the application of rules very simple as shown in section 4.

N	Symbol	Property Description
0	CS	Consonant
1	VW	Vowel
2	SV	Short Vowel
3	LV	Long Vowel
4	DP	Diphthong
5	SW	Schwa
6	VH	Vowel Height: High
7	VM	Vowel Height: Mid
8	VL	Vowel Height: Low
9	VF	Vowel Frontness: Front
10	MV	Vowel Frontness: Mid
11	VB	Vowel Frontness: Back
12	LR	Lip Rounding
13	SC	Stop Consonant
14	FC	Fricative Consonant

N	Symbol	Property Description
15	AC	Affricative Consonant
16	NC	Nasal Consonant
17	LC	Lateral Consonant
18	XC	Approximant Consonant
19	LB	Labial
20	AV	Alveolar
21	PT	Palatal
22	LD	Labio-Dental
23	DT	Dental
24	VR	Velar
25	GT	Glottal
26	VC	Consonant Voicing
27	SL	Silence
28	WB	Word Boundary

Table 1: Phoneme Properties

3. Grapheme - Phoneme Alignment

As required in the step 2 of the proposed approach, the algorithm for aligning the phonemes in the native pronunciation of a word to the graphemes in the word’s spelling is described ahead. It must be mentioned that there have been several efforts in the direction of phoneme to grapheme alignment [Lawrence and Kaye 1986, Ling and Wang, 1997; Bosch

and Daeleman, 1993; Luk and Damper, 1992]. We are using a simple heuristic technique which we find suitable for the purpose we intend to use it for.

The alignment algorithm processes a word spelling and its pronunciation in form of a phoneme vector (*PHONEME*) to produce a list of mapping between a phoneme and a string of graphemes.

We use four pseudo functions in the alignment algorithm. The arguments are the inputs to these functions and output is the returned value.

MAP(Phoneme, String of Grapheme): Inserts a node into the mapping list with the Phoneme parameter corresponding with the String of Graphemes parameter.

MATCH(Phoneme, Grapheme Position): Matches the possible graphemes of Phoneme parameters (from Appendix A) to word spelling starting at Grapheme Position. Results a zero if no match is found. If match is found, returns the length of matching grapheme string.

NEXTGR(Grapheme Position, Phoneme): If Phoneme parameter is a vowel / consonant, returns the position of grapheme ahead of Grapheme Position parameter that is / is not one of {a, e, i, o, u, y}. Also same adjacent graphemes are ignored while incrementing Grapheme Position. E.g. For the word *abhorred*, NEXTGR(4, d) returns 7 and not 5.

STR(Grapheme Position 1, Grapheme Position 2): Returns string from word spelling starting from Grapheme Position 1 to Grapheme Position 2.

The pseudo code for the alignment algorithm is given below:

- [Step 1]. Set Grapheme Position (GRP) to start of spelling
Set Phoneme Pointer (PHP) to start of phoneme vector
- [Step 2]. Insert a word boundary indicator in mapping list
i. e. MAP(\$, \$)
- [Step 3]. For all phonemes in phoneme vector:
Process [Step 4] and [Step 5]
- [Step 4]. Let L = MATCH(PHONEME[PHP], GRP)
If Match Found i.e. (L > 0)
MAP(PHONEME[PHP], STR(GRP, GRP + L - 1))
Increment PHP
Increment GRP by L
Return to [Step 3].
If Match Not Found i.e. (L = 0)
Backup Grapheme Position BGRP = GRP

GRP = NEXTGR(GRP, PHONEME[PHP])

[Step 5]. Let L = MATCH(PHONEME[PHP], GRP)

If Match Found i.e. (L > 0)

MAP(BLANK PHONEME, STR(BGRP, GRP - 1))

MAP(PHONEME[PHP], STR(GRP, GRP + L - 1))

Increment PHP

Increment GRP by L

Return to [Step 3].

If Match Not Found i.e. (L = 0)

MAP(PHONEME[PHP], BLANK STRING)

Increment PHP

Restore Grapheme Position: GRP = BGRP

Return to [Step 3].

[Step 6]. Now that all phonemes in pronunciation are processed
Map the remaining graphemes in the word spelling to a blank phoneme
MAP(BLANK PHONEME, STR(GRP, Word Length))

[Step 7]. Insert a word boundary indicator in mapping list
i. e. MAP(\$, \$)

The alignment shown in the step 2 of the example in section 2 shows the result of above algorithm in aligning graphemes to phonemes. Another example for the word *obscurely* is shown below. The native pronunciation phoneme vector consists of:

{ @ b s k y u @ l i i }

The alignment obtained by our algorithm is shown in figure 3.

\$	o	b	s	c	-	u	r	e	l	y	\$
\$	@	b	s	k	y	u@	%	l	i	i	\$

Figure 3. Grapheme - Phoneme Alignment of obscurely

4. Using the Rules

The phoneme to grapheme alignment obtained above is passed through a rule applier to add another field in the mapping list that contains the target phoneme(s) corresponding to each source phoneme in the list. The rule applier processes each node in the list except for the word boundaries and checks the applicability of each rule on that node in a sequential order. The condition for applicability of a rule on a particular node comprises of four sub-conditions. These sub-conditions use two special operators.

<PHONEME_MATCH>: Matches a phoneme expression of the type of LC, PH and RC with a phoneme value. If all the bits that ON in the phoneme value are also ON in the value of the phoneme expression, the phoneme expression and phoneme value are said to be matching. The operator can be implemented simply as a combination of bitwise AND (&) and equality (=) operators as follows:

$$((\text{Value(Phoneme Expression) } \& \text{ Phoneme Value }) = \text{Phoneme Value })$$

<GRAPHEME_MATCH>: Matches a grapheme expression of the type of GR with a string of graphemes. If one of the string of graphemes in expression is found to match, they are said to be matching. Wildcard automatically implies matching.

The sub-conditions are:

1. Rule.LC < PHONEME_MATCH > Node.Prev.PhonemeValue
2. Rule.PH < PHONEME_MATCH > Node.PhonemeValue
3. Rule.RC < PHONEME_MATCH > Node.Next.PhonemeValue
4. Rule.GR < GRAPHEME_MATCH > Node.Grapheme String

All 4 sub-conditions should be satisfied for a rule to be applicable. Once a rule is found to applicable its OPH field is copied to target phoneme field of the node being processed and processing of next node is started. The remaining rules are not tested for that particular node. In this scenario the order in which the rules are tested finds its importance.

The most general rules i.e. rules containing most of the wildcards should be placed last in the ordering of rules. The more specific rules, which are applicable in relatively fewer cases, should be tested for before the others. The ordering finds importance in the example based rule generation as explained in section 5.

5. Approaches for Building the Rules

As mentioned earlier the rules used by the rule applier in step 3 of conversion from source to target pronunciation can be built either manually or by using an example based rule generation method. Each method has its own applicability depending on the kind of rules we are building. A combination of these methods is found to be suitable.

5.1. Manual Rule Building

The rules can be written manually using the format specified in section 2.1. Then the rules are processed through a rule compiler to obtain the phoneme values of the LC, PH and RC fields automatically. Manual rule building is useful for writing the rules, which are most obviously observable.

5.2. Example Based Rule Building

The example based rule building methods works on a set of deviant words, which have pronunciation different from that generated by the available rules. The differences between

the obtained and desired phoneme vectors of are noted and accordingly the rules are generated. The rules so generated are specific to the word used for training and contains the left, right and grapheme contexts from the word itself.

After generation of rules, the rules are inserted into the rules file. It may happen that other rules generated by example-based method may be similar to a particular rule being inserted. These rules can be combined to minimize the number of rules. The condition for two rules to be merged is that the PH and OPH fields of the 2 rules should be matching. For matching PH fields the PHONEME_MATCH operator of section 4 is used. OPH fields are matched by simple string comparison. The rules can be merged by ORing the LC, PH, RC and GR fields of both the rules and as OPH is same both, it is copied into the resultant rule. The resultant rule is inserted at the top of the rule set.

The two approaches can be combined effectively. Initially the source phonemes can be mapped to the target phoneme in a one to one closest (or most frequent) match mapping function. In this case the LC, RC and GR fields will be wildcards. Then example based rule building can be applied using a set of words, which have different non-native pronunciation. The number of words in the training set to achieve satisfactory conversion is an experimental issue. The rules generated by example based approach can then finally be manually edited and extended to put in generalization, which might be observable from the examples. It must be noted that the rule applier tests the rules generated by example based methods before the manually built rules.

6. Conclusion

We have presented an approach to reuse the information available in existing Lexicons along with knowledge of pronunciation variations in non-native pronunciations in the form of rules to build non-native lexicons. The rules can be built manually or by using an example based approach. The use of grapheme context along with phoneme context in the rules allows handling sounds, which are not silent in non-native pronunciations. Considerable amount of time involved in Lexicon building can be saved using the proposed approach.

Further we are experimenting with more complex rule formats involving greater and variable window sizes in the three dimensions i.e. source phonemes, graphemes and target phonemes to be able to build more complex rules which are required in certain cases. Also we are interested in quantitatively assessing the benefits achieved through the use of these new rule formats with consideration to cost of complexity involved.

7. Appendix A Source (UK English) Phoneme Set

Phoneme	Example Words	Phoneme Properties	Possible Graphemes
uh	cup, done, bud, trouble	VW, SV, VM, VB	u, o, ou, oo, o, oe , a
e	bet, check, bed, head, any	VW, SV, VM, VF	e, a, ea
a	cat, match, bad, plaid	VW, SV, VL, VF	a, ai

o	cottage, hot, body, watch	VW, SV, VM, VB, LR	o, a, ou, au
i	bit, ship, bid, decided	VW, SV, VH, VF	i, o, a, e, y
u	pull, foot, book, should	VW, SV, VH, VB, LR	u, o, ou, oo
ii	beat, sheep, be, thief	VW, LV, VH, VF	ii, ea, ee, e, ie, ei, ey, y, i
uu	pool, boot, soup, flute, do	VW, LV, VH, VB, LR	uu, oo, u, o, ew, ou
oo	author, court, sauce, talk	VW, LV, VL, VB, LR	au, ou, aw, a, o, oo
aa	art, heart, bard, balm, aunt	VW, LV, VL, VB	a, aa, ea, au, e
@@	search, burn, bird, heard	VW, LV, VM, MV	ea, u, i, e, o, ou
ai	bite, might, like, buy, sky	VW, DP, VL, MV	i, uy, y, ai
ei	ate, mail, bay, whey, sleigh	VW, DP, VM, VF	a, ai, ay, ey, ea, ei
oi	toy, oyster, boy, groin	VW, DP, VL, VB, LR	oi, oy, o, oe, oa
au	south, how, bough	VW, DP, VL, MV, LR	au, ou, ow
ou	hole, coat, boat, no, foe	VW, DP, VM, MV	o, oa, oe, ou, ow
e@	air, bare, chair, bear, care	VW, DP, VM, VF	ai, a, ea
i@	ear, beer, here, idea, weird	VW, DP, VH, VF	ea, ee, e, ie, ei
u@	sure, jury, poor, tour	VW, DP, VL, VF, LR	u, oo, ou, u
@	about, objection, better	VW, SW, VM, MV	a, e, o, u, ou, u, i
p	pat, camper	CS, SC, LB	p
t	tap, bat, tight, typed	CS, SC, AV	t, th, d
k	camera, kill, kick, code	CS, SC, VR	c, ck, k, q, ch, kh, x
b	book, abrupt	CS, SC, LB, VC	b
d	done, bad	CS, SC, AV, VC	d
g	good, bigger, ghost	CS, SC, VR, VC	g, dh
s	sit, mass, cease, scene, six	CS, FC, AV	s, x
z	zero, quiz, boys, example	CS, FC, AV, VC	z, j, s, x
sh	ship, machine, sugar	CS, FC, PT	sh, ch, sch, s, t, c, sc
zh	vision, casual, measure	CS, FC, PT, VC	s, z, g, zh, j
f	fat, laugh, physics, cough	CS, FC, LD	f, gh, ph, p
v	various, have, of	CS, FC, LD, VC	v, f, w
th	theatre, bath	CS, FC, DT	th
dh	that, father, other	CS, FC, DT, VC	th, dh, d, t
ch	church, picture, pitch	CS, AC, PT	ch, t, tch
jh	digit, jack, judge, gym	CS, AC, PT, VC	g, j, d, jh
h	hello, loophole	CS, FC, GT	h
m	man, game, climb, hymn	CS, NC, LB, VC	m
n	man, new, none, knee	CS, NC, AV, VC	n
ng	bang, sitting, bank, uncle	CS, NC, VR, VC	ng, n
l	late, lack	CS, LC, AV, VC	l
y	yellow	CS, XC, PT, VC	y
r	reason, career, wrong	CS, XC, AV, VC	r, rh
w	water, cobweb, water, quit	CS, XC, LB, VC	w, v

References

- [Witt and Young, 1999] S M Witt and S J Young. Offline Acoustic Modeling of Non-native Accents. *Proceedings of Eurospeech99*, pages 1367 – 1370, Budapest. 1999
- [Downey and Wiseman, 1998] Simon Downey and Richard Wiseman. Dynamic and Static Improvements to Lexical Baseforms. *Proceedings of Workshop on Modeling Pronunciation Variations*, pages 157 – 162, Roldue. 1998
- [Cremelic and Martens, 1997] Nick Cremelic and Jean-Pierre Martens. Automatic Rule based Generation of Word Pronunciation Networks. *Proceedings of Eurospeech97*, pages 2459 – 2462, Rhodes. 1997
- [Amadal et al., 2000] Amadal, Korkmazshiy, and Suredan. Data Driven Pronunciation Modeling for Non-Native Speakers using Association Strength between Phones. *ASRU 2000*, volume I, pages 85 – 90. 2000
- [Fitt and Isard, 1999] Susan Fitt and Stephan Isard. Synthesis of Regional English using a Keyword Lexicon. *Proceedings of Eurospeech99*, volume II, pages 823 – 826. 1999
- [Goronzy et al., 2001] Silke Goronzy, Ralf Kompe, Stefan Rapp. Generating Non-Native Pronunciation Variants for Lexicon Adaptation. *ISCA ITRW Workshop on Adaptation Methods*, Perthshire, Scotland. 2001
- [Kishore et al., 2002] S P Kishore, Rohit Kumar, Rajeev Sangal. A Data Driven Synthesis Approach for Indian Languages using Syllable as Basic Unit. *Proceedings of Intl. Conf. on NLP 2002*, pages 311 – 316, Mumbai. 2002
- [Lawrence and Kaye, 1986] S G C Lawrence and G Kaye. Alignment of phonemes with their corresponding orthography. *Computer, Speech and Language*, pages 153 – 165. 1986
- [Ling and Wang, 1997] Charles X Ling and Handong Wang. Alignment Algorithm for Learning to Read Aloud. *Proceedings of Intl. Joint Conf. on AI*. 1997
- [Bosch and Daelemans, 1993] Antal van den Bosch and Walter Daelemans. Data Oriented methods for Grapheme to Phoneme Conversion. *Proceedings of 6th European Conference on ACL*, pages 45 – 53. 1993
- [Luk and Damper, 1992] Inference of letter – phoneme correspondences by delimiting and dynamic time warping techniques. *Proceedings of Intl. Conf. on Acoustics, Speech and Signal Processing*, 1992, volume 2, pages II.61 – II.64. 1992