TEXT TO SPEECH SYNTHESIS FOR BANGLA LANGUAGE

A thesis submitted in fulfilment of the requirements for the

Degree of

Bachelor of Science in Computer Science and Engineering,

BRAC University by

Morshed Jaman Adnan-15341010
Abir Al Quraishi- 12301052
Arifur Rahman-12201118

BRAC University, Dhaka, Bangladesh

DECLARATION

We hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This Thesis, in part has been previously submitted for degree.

Signature of the Supervisor

Abu Mohammad Hammad Ali

Signature of the Authors

………………………….......	…………………………	………………………
Morshed Jaman Adnan	Abir Al Quraishi	Arifur Rahman

ACKNOWLEDGEMENT

First of all, we would like to thank our supervisor, Mr. Abu Mohammad Hammad Ali Sir. He gave us freedom to choose the thesis topic and complete guidance throughout its development. We are lucky enough to work under his supervision. Although being extremely preoccupied with his busy schedule, he often showed much enthusiasm and took time to review our thesis work that enabled us to improve the system as well as adding new features. We learned plenty of useful things from his comments, revisions and discussions during this period.

We want to give our heartiest gratitude to Mr. Firoj Alam, former chief architect of "Kotha", a complete and first Text to Speech Synthesis for Bangla Language of CRBLP(Centre for Research on Bangla Language Processing) BRAC University for his prompt response to our problems.

ABSTARCT

In this paper, we present Text to Speech (TTS) synthesis system for the Bangla language. Here the system has been developed using phonology, G2P(Grapheme-to-Phoneme) conversion and prosodic information in the festival framework. Since Festival does not provide complete language processing support specific to various languages, it is augmented with linguistic resources to facilitate the development of TTS systems. We propose how various language processing modules such as text normalization, grapheme-to-phoneme (G2P), intonation, and duration model scan be developed and integrated within Festival to develop Bangla TTS system.

Table of contents

DECLARATION 1
ACKNOWLEDGEMENT 2
Abstract 3
CHAPTER I: Introduction 5
CHAPTER II: Text Analysis 6
2.1 Non-standard Word Analysis: 6
2.2 Token to Word Rules: 7
2.3 Number pronunciation: 10
2.4. Transliteration: 10
2.5. Multi-token numbers: 14
CHAPTER III: Lexical Analysis 15
3.1 Word pronunciation: 15
3.2 Lexicons and addenda: 15
3.3 Out of vocabulary words: 16
3.4 Building letter-to-sound (LTS) rules by hand: 17
3.5 Building (LTS) automatically: 18
3.6 Post-lexical rules: 19
CHAPTER IV: Prosodic Analysis 20
4.1 Intonation: 20
4.2 Accents/Tone Assignment: 21
4.3 F0 Contour Generation: 22
4.4 Duration: 23
4.5. Intonation pattern for Bangla: 23
CHAPTER V: Waveform Synthesis 24
5.1 Diphone database: 24
5.2 Diphone introduction: 24
5.3 Defining a Diphone list: 25
5.4 Synthesizing prompts: 27
5.5 Recording the Diphones: 30
5.6 Labeling the Diphones: 31
5.7 Extracting the pitchmarks: 32
5.8 Building LPC parameters: 33
5.9 Defining a Diphone voice: 33
5.10 Checking and correcting Diphones: 33
CHAPTER VI: Installation 34
CHAPTER VII: Future Works 36
CHAPTER VIII: Conclusion 37
APPENDICES 40

CHAPTER I: INTRODUCTION

Speech synthesis system is the automatic generation of artificial speech signal by the computer. In the last few years, this technology has been widely available for several languages for different platform ranging from personal computer to stand alone systems. The first complete Bangla Text to Speech System "Kotha" works fine while we tried to follow a different rules building the system. The necessity of human computer interactions, a Text To Speech (TTS) system for Bangla language can overcome the human computer interaction as well as the empowerment of visually impaired population and increase the possibilities of improved man-machine interaction through online newspaper reading from Internet and enhancing other information system.

CHAPTER II: TEXT ANALYSIS

2.1 Non-standard Word Analysis

We can define the task of analyzing text using a number of statistical trained models using either labeled or unlabeled text from the desired domain. Approximately it may seem to be a trivial problem, but the number of non-standard words is enough even in what is considered clean text such as news articles to make their synthesis sound bad without it. Full Non Standard Words (NSW) modeling description and justification to be added later.

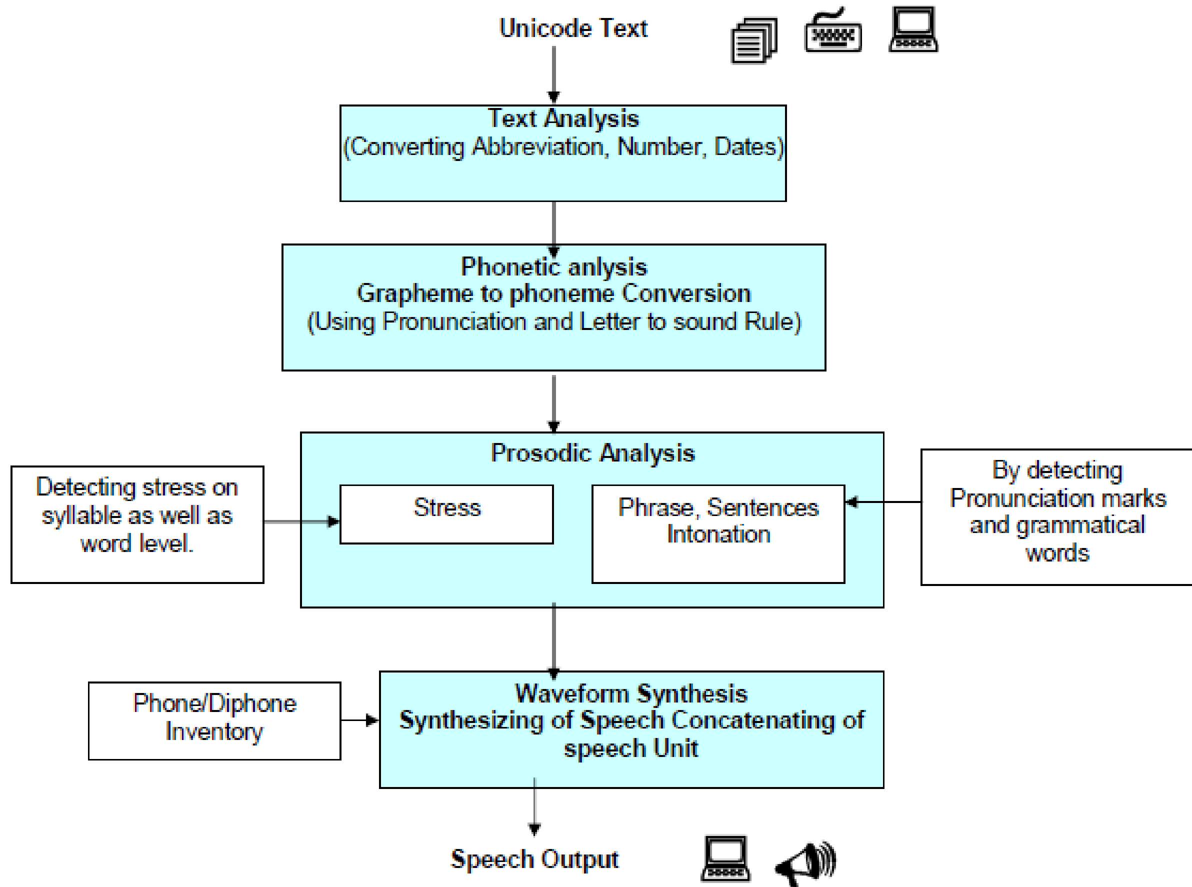

Fig 1: Bangla Text To Speech Block Diagram

2.2 Token to Word Rules

The basic model in Festival is that each token will be mapped to a list of words by a call to a token_to_word function. This function will be called on each token and it should return a list of words. It may check the tokens to context (within the current utterance) too if necessary. The default action should (for most languages) simply be returning the token itself as a list of own word (itself). For example our basic function should look like

```
(define (bangla_token_to_words token name)
"(bangla_token_to_words TOKEN NAME)

It returns a list of words for the NAME from TOKEN. This primarily allows the treatment of numbers, money etc."

(cond
  ((string-matches name "[0-9]+") ;; any string of digits
      (mapcar
          (lambda (d)
             (car (cdr (assoc_string d bangla_digit_names))))
          (symbolexplode name)))
  (t
  (list name))
```

This function is to be set in our voice selection function as the function for token analysis (set! token_to_wordsbangla_token_to_words). This function is added to deal with all tokens that are not in our lexicon cannot be treated by our letter-to-sound rules, or are ambiguous in some way and require context to resolve. For example suppose we wish to simply treat all tokens consisting of strings of digits to be pronounced as a string of digits (rather than numbers).We would add something like the following

```
(set! bangla_digit_names'

((0 "zero")
(1 "one")
………..
```

………..
………..
(3 "three")
(4 "four")
(5 "five")
(6 "six")
(7 "seven")
(8 "eight")
(9 "nine")))

But more elaborate rules are also necessary. Some tokens require context to disambiguate and sometimes multiple tokens are really one object e.g. “$12 billion” must be rendered as “twelve billion dollars”, where the money name crosses over the second word. Such multi-token rules must be split into multiple conditions, one for each part of the combined token. Thus we need to identify the “$DIGITS” is in a context followed by “?illion” (million, billion, trillion). The code below renders the full phrase for the dollar amount. The second condition ensures nothing is returned for the “?illion” word as it has already been dealt with by the previous token.

```
((and (string-matches name "\\$[123456789]+")

      (string-matches (item.feat token "n.name") ".*illion.?"))

            (append
               (digits_to_cardinal (string-after name "$")) ;; amount

            (list (item.feat token "n.name")) ;; magnitude

(list "dollars"))) ;; currency name

((and (string-matches name ".*illion.?")(string-matches (item.feat token "p.name") "\\$[123456789]+"));; dealt with in previous tokennil)
```

Note that this still is not enough as there may be other types of currency pounds, yen, francs etc, some of which may be mass nouns and require no plural (e.g. “yen}” and some of which make be count nouns require plurals. Also this only deals with whole numbers of *illions, “$1.25 million” is common too.

These tokens can be defined within

festival/lib/token.scm

.

A large list of rules is typically required. They should be looked upon as breaking down the problem into smaller parts, potentially recursive. For example hyphenated tokens can be split into two words. It is probably wise to explicitly deal with all tokens than are not purely alphabetic. Maybe we can have a catch function that spells out all tokens that are not explicitly dealt with (e.g. the numbers). For example we could add the following as the penultimate condition in our token_to_word function

```
((not (string-matches name "[A-Za-z]"))
(symbolexplode name))
```

Note this isn't necessarily correct when certain letters may be homographs. For example the token "a" may be a determiner or a letter of the alphabet. When it's a determiner it may (often) be reduced while as a letter it probably isn't (i.e. pronunciation in "@" or "ei".

Other languages also show this problem (e.g. Spanish "y"). Therefore, when we call symbol explode we don't want just the letter but to also specify that it is the letter pronunciation we want and not the any other form. To ensure the lexicon system gets the right pronunciation we there wish to specify the part of speech with the letter. Actually rather than just a string of atomic words being returned by the token_to_word function the words may be descriptions including features. Thus for example we don't just want to return

```
(a b c)
```

We want to be more specific and return

```
(((name a) (posnn))
((name b) (posnn))
((name c) (posnn)))
```

This can be done by the code

```
((not (string-matches name "[A-Za-z]"))
```

```
(mapcar
  (lambda (l)
    ((list 'name l) (list 'pos 'nn)))
  (symbolexplode name)))
```

The above assumes that all single characters symbols (letters, digits, punctuation and other "funny" characters have an entry in our lexicon with a part of speech field nn, with a renunciation of the character in isolation.

2.3 Number pronunciation

Almost everyone will expect a synthesizer to be able to speech numbers. As it is not feasible to list all possible digit strings in our lexicon we will need to provide a function that returns a string of words for a given string of digits. In its simplest form we provide a function that decodes the string of digits.

2.4 Transliteration

The first step of our text analysis is transliteration based on our phoneme set. Our entire phoneme set and their features are given in table 1 and 2. This phoneme set and their features values are used as the input of the festival for our language. We transliterate our Unicode script into ASCII coded English script. The transliteration table 3 and 5 is given in the next consecutive table.

Consonants/e¨ÄbeY

Manner of articulation		Place of articulation										
		Velar		Palato-alveolar		Alveolar		Dental		Bilabial		Glottal
		In-aspirate – Aíciv	Aspirate- gnciv	In-aspirate – Aíciv	Aspirate- gnciv	In-aspirate - Aíciv	Aspirate- gnciv	In-aspirate - Aíciv	Aspirate- gnciv	In-aspirate – Aíciv	Aspirate- gnciv	
Plossive/ Stop	Voiceless	K/ k	L/ k^h	P/ c	Q/ c^h	U/ τ	V/ τ□	Z/ τ∉	_/ τ∉□	c/ π	d/ π□	
	Voiced	M/ g	N/ g^h	R/ □	S/ □	W/ δ	X/ δ□	`/ δ∉	a/ δ∉□	e/ β	f/ β□	
Fricative	Voiceless			k,l /Σ		m/ σ						t
	Voiced											n /η
Nasal	Voiced	0 s/ N		T/ ϑ		b/ ν, Y/ ν				g/μ		
Liquid/ Lateral	Voiced			h/ □		j/ λ						
Trill	Voiced					i/ ρ						
Flapped	Voiced					o/ 4	p/ 4					
Glide	Voiceless			q/ φ								
	Voiced									e/ ω		

Table 1 : Bangla Consonant phoneset

Vowels/¯ieY

A/O /	A/ a /	B-C/ i /	D-E/ u /	F/ ri /
G/e /	H/ oi /	I/ o /	J/ ou /	

Vowel		A/O /	A/ a /	B-C/ I /	D-E/ u /	F/ ri /	G/e /	H/ oi /	I/ o /	J/ ou /	G˜/ { /
Height	high	-	-	+	+		-		-	-	-
	Low	+	+	-	-		-		-	+	+
front ness	Front			+			+				+
	Back	+	+		+				+	+	
Lip rounding		-/+	-	-	+		-		+	+	-
Vowel_t	Tense	-	+	+	+		+		-	-	+
Length	diphthong							+		+	
	Long			-+	-+						

Table 2 : Bangla vowel phoneset

Vowels

Letter	Transliteration
□	a
□	aa
□	i
□	ii
□	u
□	uu
□	ri
□	e
□	oi
□	o
□	ou

Table 3: Vowel Mapping

Modifiers **

Symbol with [k□] (□(	Name	Function	Transliteration
□□	hôshonto	Suppresses the inherent vowel	-
□□□	khônđo tô	Final unaspirated dental [t] (□)	t1
□□	Ônushshôr	Final velar nasal	Ng
□□	Bishôrgo	Adds voiceless breath after vowel	:
□□	Chôndrobindu	Nasalises vowel	n1

Table 4 : Bangla transliteration table on modifiers

** In our implementation we omitted these modifiers.

Consonants

Letter	Transliteration
□	k
□	kh
□	g
□	gh
□	ng
□	c
□	ch
□	j
□	jh
□	nio
□	t
□	th
□	d
□	dh
□	n
□	ta
□	to
□	da
□	dh
□	n
□	p
□	ph
□	b
□	bh
□	m
□	z
□	r

Consonants

Letter	Transliteration
□	l
□	sh
□	sh
□	s
□	h
□□	y
□□	ra
□□	Rh

Table 5: Consonant Mapping

2.5 Multi-token numbers

A number of languages use spaces within numbers where English might use commas. For example German, Polish and others text may contain64 000 to denote sixty four thousand. As this will be multiple tokens in Festival's basic analysis it is necessary to write multiple conditions in our token_to_word function.

CHAPTER III: LEXICAL ANALYSIS

3.1 Word pronunciations

The lexicon structure that is basically available in Festival takes both a word and a part of speech (and arbitrary token) to find the given pronunciation. For English this is probably the optimal form and since we map Bangla Unicode to English ASCII code this is also optimal for our approach. Although there are homographs in the language, the word itself and a fairly broad part of speech tag will mostly identify the proper pronunciation. An example entry is

(“আপনি”)
("aapni" n (((aa p) 0) ((n i) 0)))

In some languages lexicon is fully predictable, in other words highly irregular. In some cases this field may be more appropriately used for other purpose, e.g. tone type in Chinese.

3.2 Lexicons and addenda

The basic assumption in Festival is that we will have a large lexicon, tens of thousands of entries that are used as a standard part of an implementation of a voice. Letter-to-sound rules are used as back up when a word is not explicitly listed. This view is based on how English is best dealt with. However this is a very flexible view, an explicit lexicon isn’t necessary in Festival and it may be possible to do much of the work with letter-to-sound rules. However, even when there is strong relationship between the letters in a word and their pronunciation we still find a lexicon useful.

In addition to a large lexicon Festival also supports a smaller list called addenda which is primarily provided to allow specific applications and users to add entries that aren’t in the existing lexicon.

3.3 Out of vocabulary words

No matter what we do we must provide something even if it is simply replacing the unknown word with the word "unknown" (or its local language equivalent). By default a lexicon in Festival will throw an error if a requested word isn't found. To change this we set the lts_method. Most usefully we reset this to the name of function, which takes a word and a part of speech specification and returns a word pronunciation as described above. For example, we are always going to return the word unknown but print a warning that the word is being ignored. A suitable function is

```
(define
    (mylex_lts_function word feats);;"Deal with out of vocabulary word."

    (format t "unknown word: %s\n" word)
 ("unknown" n (((uh n) 1) ((n ou n) 1)))))
```

Note the pronunciation of "unknown" must be in the appropriate phoneme set. Also the syllabic structure is required. We need to specify this function for our lexicon as follows

```
(lex.set.lts.method
'mylex_lts_function)
```

At one level above merely identifying out of vocabulary words, they can be spelled, this of course isn't ideal but it will allow the basic information to be passed over to the listener. This is done with the out of vocabulary function, as follows.

```
(define
 (mylex_lts_function word feats)

"Deal with out of vocabulary words by spelling out the letters in the word."

(if
   (equal? 1 (length word))
        (begin
           (format t "the character %s is missing from the lexicon\" word)
'           ("unknown" n (((uh n) 1) ((n ou n) 1))))
```

```
(cons word 'n
   (apply
      append
     (mapcar
       (lambda (letter)
          (car (cdr (cdr (lex.lookup letter 'n)))))
(symbolexplode word))))))
```

A few points are worth noting in this function. This recursively calls the lexical lookup.

3.4 Building letter-to-sound rules by hand

In Festival there is a letter to sound rule system that allows rules to be written, but we also provide a method for building rule sets automatically which will often be more useful. The choice of using hand-written or automatically trained rules depends on the language we are dealing with and the relationship it has between its orthography and its phoneme set.

Handwritten letter to sound rules are context dependent and rewrite rules which are applied in sequence mapping strings letters to string of phones (though the system does not explicitly care what the types of the strings actually will be used for. The basic form of the rules is

(LC [alpha] RC => beta)

Which interprets as alpha, a string of one or more symbols on the input tape is written to beta, a string of zero or more symbols on the output tape, when in the presence of LC, a left context of zero or more input symbols, and RC a right context on zero or more input symbols. Note the input tape and the output tape are different, although the input and output alphabets need not be distinct the left hand side of a rule only can refer to the input tape and never to anything has that been produce by a right hand side. Thus rules within a rule set cannot "feed" or "bleed" themselves. It is possible to cascade multiple rule sets. For example to deal with the pronunciation of the letter (চ)"ch" word initially in Bangla we may right two rules like

(# [c h] r => k)
(# [c h] =>ch)

3.5 Building letter-to-sound rules automatically

The process described here is not (yet) fully automatic but the hand intervention required is small and may easily be done even by people with only a very little knowledge of the language being dealt with. The process involves the following steps

- Pre-processing lexicon into suitable training set
- Defining the set of allowable pairing of letters to phones. (We intend to do this fully automatically in future versions).
- Constructing the probabilities of each letter/phone pair.
- Aligning letters to an equal set of phones/_epsilons_.
- Extracting the data by letter suitable for training.
- Building CART models for predicting phone from letters (and context).
- Building additional lexical stress assignment model (if necessary).

All except the first two stages of this are fully automatic. Before building a model it's wise to think a little about what we want it to do. Ideally the model is an auxiliary to the lexicon so only words not found in the lexicon will require use of the letter-to-sound rules. Thus only unusual forms are likely to require the rules; more precisely the most common words, often having the most.

It is best to split the data into a training set and a test set to know how well the training has worked. In our tests we remove every tenth entry and put it in a test set.

This can sometimes be initially done by hand then checked against the training set. Initially construct a file of the form

```
(require 'lts_build)
(set! allowables
  '((a _epsilon_)
    (b _epsilon_)
    (c _epsilon_)
     ...
    (y _epsilon_)
    (z _epsilon_)
(# #)))
```

All letters that appear in the alphabet should (at least) map to _epsilon_, including any accented characters that appear in that language. Note the last two hashes are used to denote the beginning and end of word and are automatically added during training, they must appear in the list and should only map to themselves.

To incrementally add to this allowable list run festival as festival allowables.scm and at the prompt type

```
festival> (cummulate-pairs "oald.train")
```

3.6 Post-lexical rules

Post lexical rules are general set of rules which modify the segment relation (or any other part of the utterance for that matter), after the basic pronunciations have been found. In Festival post-lexical rules are defined as functions which will be applied to the utterance after intonation accents have been assigned. A Scheme function that implements as follows

```
(define (plr_rp_final_rutt)
   (mapcar
    (lambda (s)
      (if (and (string-equal "r" (item.name s)) ;; this is an r
      ;; it is syllable final
        (string-equal "1" (item.feat s "syl_final"))
        ;; the syllable is word final
        (not (string-equal "0"
          (item.feat s "R:SylStructure.parent.syl_break")))
          ;; The next segment is not a vowel
        (string-equal "-" (item.feat s "n.ph_vc")))
      (item.delete s)))
(utt.relation.itemsutt 'Segment)))
```

In Bangla we also use post-lexical rules for phenomena such as vowel reduction.

CHAPTER IV: PROSODIC ANALYSIS

Festival basically supports two methods for predicting prosodic phrases, though any other method can easily be used. Note that these do not necessary entail pauses in the synthesized output. Pauses are further predicted from prosodic phrase information. The first basic method is by CART tree. A test is made on each word to predict it is at the end of a prosodic phrase. The basic CART tree returns B or BB (though may return what we consider is appropriate form break labels as long as the rest of our models support it). The two levels identify different levels of break, BB being a used to denote a bigger break (and end of utterance).

The following tree is very simple and simply adds a break after the last word of a token that has following punctuation. Note the first condition is done by a lisp function as we want to ensure that only the last word in a token gets the break.

```
(set! simple_phrase_cart_tree
'
  ((lisp_token_end_punc in ("?" "." ":"))
  ((BB))
  ((lisp_token_end_punc in ("'" "\"" "," ";"))
  ((B))
  ((n.name is 0) ;; end of utterance
   ((BB))
  ((NB))))))
```

This CART tree is defined festival/lib/phrase.scm in the standard distribution and is certainly a good first step in defining a phrasing model for a new language.

4.1 Intonation:

There are probably more theories of intonation than there are people working in the field. Traditionally speech synthesizers have had no intonation models (just a monotone) or very poor ones. But today the models are becoming quite sophisticated such that much of the intonation tunes produced are often very reasonable. Intonation prediction can be split into two tasks:

Accents: (and/or tones) this is done on a per syllable basis, identifying which syllables are to be accented as well as what type of accent is required (if appropriate for the theory).

F0 contour: Given the accents/tones generate an F0 contour. This must be split into two tasks as it is necessary for duration prediction to have information about accent placement, but F0 prediction cannot take place until actual durations are known. Vowel reduction is another example of something, which should come between the two parts of tune realization.

4.2 Accents/Tone Assignment:

Accent is a property of a word in context - it is a way to mark intonation prominence in order to 'highlight' important words in the speech. Syllabic pitches movements are represented by three types of syllabic intonations namely rise, fall and flat. In our Bangla language we have three types intonation pattern also called accent types. Boundary tones are the intonation events that occur at the end (or start) of prosodic phrases. The classic examples are final rises (sometimes used in questions) and falls (often used in declaratives). Festival allows accent placement by decision tree. In our implementation a much simpler example (which is defaults in festival) is just to have accents on stressed syllables in content words (and single-syllable content words with no stress). A decision tree to do this is as follows

```
(set! simple_accent_cart_tree
   ((R:SylStructure.parent.gpos is content)
    ((stress is 1)
     ((Accented))
     ((position_type is single)
      ((Accented))
      ((NONE))))
((NONE))))
```

The above tree simply distinguishes accented syllables from non-accented. This simpler solution, such as fixed prosody, or fixed declination, is the most basic for voice but apart from debugging a voice is simpler than required. But to do this in

Festival, we need a CART tree to predict accent, and rules to add the accent. Festival wagon tools can be used to predict accents, and similar tree should be used to predict boundary tones as well.

4.3 F0 Contour Generation:

The F0 is the basic tune in speech for males it usually ranges between about 90Hz and 120Hz and about 140Hz to 280Hz in females. In general an F0 starts higher at the beginning on a sentence and gets lower of time, reflect the depletion in the air rate as we speak, though it is possible to make it rise over time. It is obvious that accent position influences durations and an F0 contour cannot be generate without knowing the durations of the segments the contour generate over. Festival supports a number of methods, which allow generation of target F0 points. These target points are later interpolated to form an F0 contour. The simplest model adds a fixed declining line. This is useful when intonation is ignored in order test other parts of the synthesis process. The General Intonation method allows a Lisp function to be written which specify a list of target points for each syllable. This is powerful enough to implement many simple and quite powerful theories. The following function returns three target points for accented syllables given a simple pattern for accents syllables.

```
(define (targ_func1 utt syl)
        (targ_func1 UTT ITEM) ;;Returns a list of targets for the given syllable."
         (let ((start (item.feat syl "syllable_start"))
               (end (item.feat syl "syllable_end")))
         (if (not (eq? 0 (length (item.relation.daughters syl "Intonation")))
        (list
          (list start 110)
          (list (/ (+ start end) 2.0) 140)
        (list end 100)))))
```

Of course this is too simple. Declination, changes in relative heights, speaker parameterization are all really required. In our implementation, we used this simple method for F0. For larger context we have to think about ToBI (Tones and Break Indices) is an intonation based labeling, standard for larger speech databases.

4.4 Duration:

There is lot of methods available in festival for explicit segmental durations are necessary for synthesis. The easiest method is to use a fixed size for all phones (e.g.100 millisecond for each phone). The next simplest model assigns the average duration for that phone (from some training data). This actually produces reasonable results. This simple method is used in our implementation. We used fixed durations for each phones, though there is no logical reason why we used these fixed value. This is taken from Kiswahili TTS system.

```
(set! bu_bangla_adnan::phone_durs
'(
;;; PHONE DATA
;; name zero mean in seconds e.g.
; all phones on bu_bangla_adnan
(# 0.0 0.250)
(SIL 0.0 0.250)
(a 0.0 0.100)
(aa 0.0 0.100)
(i 0.0 0.100)
(ii 0.0 0.100)
…………….
…………....))
```

4.5 Intonation pattern for Bangla

We have some intonation rule in Bangla language. This has been considered that the syllables must have either rising, falling or flat intonation and as the word comprised of number of syllables, their intonation pattern would be a combination of series of rising, falling and flat intonations. The syllabic stylized version is thus gets a RFF1 (Rise, Flat, fall) intonation pattern. Highest probability of occurrence of intonation pattern for different syllabic words has been considered to frame the intonation rule. As we know that Bengali is a bound stress language; so every word normally starts with a rising intonation pattern.

CHAPTER V: WAVEFORM SYNTHESIS

This is one of the major parts in TTS. Several techniques are available for waveform synthesis, articulator synthesis, formant synthesis, and concatenative synthesis. The techniques described in festival are concatenative synthesis. Concatenative synthesis techniques not only give the most natural sounding speech synthesis. Two techniques are available in concatenative synthesis diphone, unit selection. We used diphone database for waveform synthesis. Recording is one of the big issues in these techniques. So, professional speaker should record our voices in a clean environment. In this chapter we present voice building process that we followed step by step.

5.1 Diphone databases

This following section describes the processes involved in designing, listing recording, and labeling a diphone database for a language.

5.2 Diphone introduction

The basic idea behind building diphone databases is to explicitly list all possible phone-phone transitions in a language. This makes the incorrect but practical and simplifying assumption that co-articulator effects never go over more than two phonemes. The exact definition of phonemes here is in general nontrivial, and what a "standard" phone set should be is not uncontroversial -- various allophonic variations, such as light and dark /l/, may also be included. Unlike generalized unit selection where multiple occurrences of phones may exists with various distinguishing features, in a diphone database only one occurrence of each diphone is recorded. This makes selection much easier but also makes for a large laborious collection task. In general, the number of diphones in a language is the square of the number of phones. However, in natural human languages, there are phonotactic constraints -- some phone-phone pairs, even whole classes of phones-phone combinations, may not occur at all. These gaps are common in the world's languages. The exact definition of *never exist* is also problematic. Humans can often generate those so-called non-existent diphones if they try, and one must always think about phone pairs that cross over word boundaries as well, but even

then, certain combinations cannot exist; for example, (ং)/ng/ (ং)/ng/ in Bangla is probably impossible (we would probably insert a silence). (ং)/ng/ may really only appears after the vowel in a syllable (in coda position); however, in other languages it can appear in syllable-initial position. /hh/ cannot appear at the end of a syllable, though sometimes it may be pronounced when trying to add aspiration to open vowels. Diphone synthesis, and more generally any concatenative synthesis method, makes an absolutely fixed choice about which units exist, and in circumstances where something else is required, a mapping is necessary. Duplicating all the vowels (e.g. stressed/unstressed versions) will significantly increase the database size. These inventory questions are open, and depending on the resources we are willing or able to devote, can be extended considerably. It should be clear, however, that such a list is simply a basic set.

5.3 Defining a Diphone list

Since diphones need to be clearly articulated, various techniques have been proposed to elicit them from subjects. One technique is to use target words embedded carrier sentences to ensure that the diphones are pronounced with acceptable duration and prosody (i.e. consistently). We have typically used nonsense words that iterate through all possible combinations; the advantage of this is that we don't need to search for natural examples that have the desired diaphone, the list can be more easily checked and the presentation is less prone to pronunciation errors than if real words were presented. The words look unnatural but collecting all diaphones in not a particularly natural thing to do. For best results, we believe the words should be pronounced with consistent vocal effort, with as little prosodic variation as possible.

Some example code is given in src/diphone/darpaschema.scm. The basic idea is to define classes of diphones, for example: vowel-consonant, consonant-vowel and consonant-consonant. Then define carrier contexts for these and list the cases. Here we use Festival's Scheme interpreter to generate the list though any scripting language is suitable. Our intention is that the diphone will come from a middle syllable of the nonsense word so it is fully articulated and minimize the articulator

effects at the start and end of the word. For example to generate all vowel-vowel diphone we define a carrier

```
(set! vv-carrier ’((# t aa t) (t aa #))) ;;”((ত আ ত)(ত আ))”
```

And we define a simple function that will enumerate all vowel vowel transitions

```
(define (list-vvs)
  (apply
    append
    (mapcar
    (lambda (v1)
      (mapcar
      (lambda (v2)
        (list
          (string-append v1 "-" v2)
      (append (car vv-carrier) (list v1 v2) (car (cdr vv-carrier)))))
    vowels))
vowels)))
```

The actual Lisp code returns a list of diphone names and phone string. To be more efficient, the DARPAbet example produces consonant-vowel and vowel-consonant diphones in the same nonsense word, which reduces the number of words to be spoken quite significantly. Our voice talent will appreciate this. Although the idea seems simple to begin with, simply listing all contexts and pairs, there are other constraints. Some consonants can only appear in the onset of a syllable (before the vowel), and others are restricted to the coda. While one can collect all the diphones without considering where they fall in a syllable, it often makes sense to collect diphones in different syllabic contexts. Consonant clusters are the obvious next set to consider; thus the example DARPAbet schema includes simple consonant clusters with explicit syllable boundaries.

A second and related problem is language interference, which can cause phoneme crossover. Because of the prevalence of English, especially in electronic text, how many "foreign" phones should be considered for addition is a research issue.

However, we choose to construct the diphone list, and whatever examples we choose to include, the tools and scripts included with this document require that it be in a particular format. Each line should contain a file id, a prompt, and a diphone name (or list of names if more than one diphone is being extracted from

that file). The file id is used to in the filename for the waveform, label file, and any other parameters files associated with the nonsense word. We usually make this distinct for the particular speaker we are going to record, e.g. their initials and possibly the language they are speaking. The prompt is presented to the speaker at recording time, and here it contains a string of the phones in the nonsense word from which the diphones will be extracted. For example the following is taken from our generated nonsense word list.

bangladiph.list contains nonsense word list

(bangla_0001 "# t aa k a k aa #" ("k-a" "a-k"))
(bangla_0002 "# t aakh a khaa #" ("kh-a" "a-kh"))
(bangla_0003 "# t aa g a g aa #" ("g-a" "a-g"))
(bangla_0004 "# t aagh a ghaa #" ("gh-a" "a-gh"))
(bangla_0005 "# t aa c a c aa #" ("c-a" "a-c"))
(bangla_0006 "# t aach a chaa #" ("ch-a" "a-ch"))
(bangla_0007 "# t aa j a j aa #" ("j-a" "a-j"))
(bangla_0008 "# t aajh a jhaa #" ("jh-a" "a-jh"))
(bangla_0009 "# t aa t a t aa #" ("t-a" "a-t"))
(bangla_0010 "# t aath a thaa #" ("th-a" "a-th"))
....
....

Note the explicit syllable boundary marking - for the consonant-consonant diphones is used to distinguish them from the consonant cluster examples that appear later.

5.4 Synthesizing prompts

To help keep pronunciation consistent we synthesize prompts and play them to our own voice talent at collection time. This helps the speaker in two ways -- if they mimic the prompt they are more likely to keep a fixed prosodic style; it also reduces the number of errors where the speaker vocalizes the wrong diphone. Of course for new languages where a set of diphones doesn’t already exist, producing prompts is not easy, however giving approximations with diphones from other languages may work. The problem then is that in producing prompts from a different phoneset, the speaker is likely to mimic the prompts hence the diphone

set will probably seem to have a foreign pronunciation, especially for vowels. Furthermore, mimicking the synthesizer too closely can remove some of the speaker's natural voice quality, which is under their (possibly subconscious) control to some degree. Even when synthesizing prompts from an existing diphone set, one must be aware that that diphone set may contain errors or those certain examples will not be synthesized appropriately. Because of this, it is still worthwhile monitoring the speaker to ensure they say things correctly. The basic code for generating the prompts is in src/diphone/diphlist.scm. The prompts can be generated from the diphone list as described above (or at the same time). The example code produces the prompts and diphone labels files which can be used by the aligning tool described below. Before synthesizing, the function Diphone_Prompt_Setup is called within the bd_schema.scm file. We defined this to set up the appropriate voices in Festival, as well as any other initialization we might need -- for example, setting the fundamental frequency (F0) for the prompts that are to be delivered in a monotone (disregarding so-called micro prosody, which is another matter). This value is set through the variable FP_F0 and should be near the middle of the range for the speaker, or at least somewhere comfortable to deliver. For our system we have defined it the following way.

```
(define
(Diphone_Prompt_Setup)
(Diphone_Prompt_Setup)
```

It is called before synthesizing the prompted waveforms.

```
(set! FP_F0 90) ;; lower F0 than ked)
```

It is a good idea to map phoneset into one existing phoneset which can be used to generate the prompt files within the prompt-wav folder. We mapped our Bangla phoneset into MRPA phone set in method bd2us within the bd_schema.scm file.

```
(set! bd2us_map
'((a a)
(aaaa)
    (i ii)
    (u uu)
    …….
```

```
………
(iu ii uu)
(io ii ou)
(ia ii aa)
(uiuu ii)
..........
..........)
```

Phones that are not explicitly mentioned map to themselves (e.g. most of the consonants). Finally, we define Diphone_Prompt_Word to actually do the mapping. We call the previously mentioned bd2us method within this function. where the mapping involves more than one MRPA phone we add an extra segment to the Segment (defined in the Festival manual) relation and split the duration equally between them. The basic function looks like

```
(define
(Diphone_Prompt_Wordutt)
(Diphone_Prompt_Wordutt)
;; Specify specific modifications of the utterance before synthesis specific to this
particular phoneset.

(mapcar
  (lambda (s)
    (let ((n (item.name s))
    (newn (cdr (assoc_string (item.name s) bd2us_map))))
(cond
   ((cdr new n) ;; its a dual one
     (let ((new i (item.insert s (list (car (cdr new n))) 'after)))
       (item.set_featnewi "end" (item.feat s "end"))
       (item.set_feat s "end"
         (/ (+ (item.feat s "segment_start")
         (item.feat s "end"))
       2))
     (item.set_name s (car newn))))
    (new n
    (item.set_name s (car newn)))
(t
;; as is
))))
(utt.relation.itemsutt 'Segment))utt)
```

By convention, the prompt waveforms are saved in prompt-wav/, and their labels in prompt-lab/. The prompts may be generated after the diphone list is given using the following command:

festivalfestvox/bd_schema.scmfestvox/diphlist.scm
festival> (diphone-gen-schema "bangla" "etc/bangladiph.list")

Once we have thediphone list schema generated in the fileetc/bangladiph.list, we do the following

festivalfestvox/bd_schema.scmfestvox/diphlist.scm
festival> (diphone-gen-waves "prompt-wav" "prompt-lab" "etc/bangladiph.list")

5.5 Recording the Diphones

After generating the prompt-wav files we move into the recording part. We used to Festival systems provide recording method for our recording purpose. Recording each of the 2517 generated nonsense words takes nearly 3 seconds. Although it takes some getting used to since the prompt files are uttered using us_rab_diphone voice while we had to adjust them according to our Bangla phoneset. The recordings can be done using the following command.

bin/prompt_themetc/bangladiph.list

Since we had to recording in several sessions we used the index number of the prompt at the end of the command. For example, in order to start recording from bangla_0101 we have to run

bin/prompt_themetc/bangladiph.list 0101

The prompts occur one at a time with few seconds time gap until the last one in our case to 2517th prompt.

5.6 Labeling the Diphones

Once the recording is complete it is time to label the wav files. The label files are created and saved into lab folder as .lab files. The labeling can be done outside of Festival system with other tools. We used the system provided tool for labeling purpose. This is an auto labeling process and some of the labels were missing which had to be manually.

To run the script over the prompt waveforms

bin/make_labs prompt-wav/*.wav

Of course the labeling can be done for individual .wav files. For example the following command only labels the bangla_0101.wav file.

bin/make_labs prompt-wav/bangla_0101.wav

Once the nonsense words have been labeled, we need to build diphone index file. The index identifies which diphone comes from which files, and their start, middle and end point. This can be automatically built from the label files (mostly). The Festival script festvox/src/diphones/make_diph_index will take the diphone list (as used above), find the occurrence of each diphone in the label files and build an index. The index consists of a simple header, followed by a single line for each diphone: the diphone name, the field, start time, mid-point (i.e. the phone boundary) and end time. The times are given in seconds..

An example from the start of a diphone index file is
EST_File index
DataType ascii
NumEntries 2517
IndexName bu_bangla_diphone
EST_Header_End
………………………………...
…………………………………
n-# bangla_2475 0.59500003 0.68000001 0.73000002
dh-# bangla_2474 0.65499997 0.73000002 1.9450001
d-# bangla_2473 0.61000001 0.69 0.75
th-# bangla_2472 1.8200001 1.88 1.9299999
t-# bangla_2471 1.8150001 1.885 1.9400001
……………………………………………..

Note the number of entries field must be correct; if it is too small it will (often confusingly) ignore the entries after that point.

This file can be created with a diphone list file and the lab files in by the command

bin/make_diph_indexetc/bangladiph.listdic/bangladiph.est

Some of the entries in the index may result in 0 for start, middle and end time which will have to be corrected manually.

5.7 Extracting the Pitchmarks

Upon generating the index file the pitchmarks can be extracted using the index file. The pitchmark files are saved within the pm folder. The following command is used extract the pitchmarks.

bin/make_pm_wave wav/*.wav

A program to move the predicted pitchmarks to the nearest peak in the waveform is also provided. This is almost always a good idea, even for EGG extracted pitchmarks

bin/make_pm_fix pm/*.pm

A table of power modifiers for each file can be calculated by

bin/find_powerfactors lab/*.lab

Then we build the pitch-synchronous LPC coefficients, which used the power factors ifthey've been calculated.

bin/make_lpc wav/*.wav

5.8 Building LPC parameters

Then we build the pitch-synchronous LPC coefficients, which used the power factors if they've been calculated.

```
bin/make_lpc wav/*.wav
```

5.9 Defining a Diphone voice

Once we are happy with the completed voice we can package it for distribution. The first stage is to generate a group file for the diphone database. This extracts the subparts of the nonsense words and puts them into a single file offering something smaller and quicker to access. The group file can be built as follows.

```
festivalfestvox/bu_bangla_adnan_diphone.scm
"(voice_bu_bangla_adnan_diphone)"
...
festival (us_make_group_file "group/adnanlpc.group" nil)
```

The us_ in the function names stands for UniSyn (the unit concatenation subsystem in Festival) and nothing to do with US English.

5.10 Checking and correcting Diphones

This is the part where we verify the diphone pronunciation through running different diphones using Say_Phones command within festival. Then we check the lexicon file for lts rules and token files as well. It is evident that most systems build may not run in the first attempt. We corrected some diphones as well as some entries which are not mapped within our schema files.

CHAPTER VI: INSTALLATION

First we need to download all the files from the following locations

http://festvox.org/packed/festival/2.4

http://festvox.org/festvox-2.7

Then we set up the environment variables for voice building

```
export ESTDIR=`pwd`/speech_tools
export FESTVOXDIR=`pwd`/festvox
export FLITEDIR=`pwd`/flite
export SPTKDIR=`pwd`/SPTK

mkdir SPTK
patch -p0 <festvox/src/clustergen/SPTK-3.6.patch
```

Building SPTK

```
./configure --prefix=$SPTKDIR
make
make install
```

Building speech tool

```
./configure
make
make test
```

Building festival

```
./configure
make
make test
```

Exporting path to festival binary folder

```
#>export PATH=$PATH:/home/build/festival/bin
```

Testing installation

```
#>festival
```

This command should open the festival command line interpreter.

```
festival>
```

Then testing with default voice

```
festival>(SayText “hello world”)
```

At this point we can assume that the system is ready for building our own voice.

CHAPTER VII: FUTURE WORKS

We are in preliminary stage in our implementation. Lots of issue undiscovered now. We have to work out on the following issues.

1. Text Analysis: Text normalization using scheme for larger context.
2. Phonetic Analysis:
   i. Automatic lexicon entries instead of adding manually.
   ii. Find out LTS or G2P rule
3. Prosody Analysis: Have to build CART tree for Bangla intonation pattern using ToBI(Tone and Break Indices) or wagon.
4. Waveform synthesis:
   i. We have to overcome the limitation of multisyn unit selection technique.
   ii. Plan to build diphone database for better sound quality. This is one of the huge research issues.

CHAPTER VIII: CONCLUSIONS

The described speech synthesis system is a preliminary TTS system for the Bangla language. The synthetic speech produced by the system is not yet intelligible, but nearer to naturalness. Improvement of intelligibility and naturalness depend on proper works in different context. Preparation of the diphone inventory is laborious and time-consuming, but it can produce batter quality sound, which is proved in different language. Here the whole process done by diphone database technique. The Bangla TTS system is very much important for the visually impaired people of our country, who cannot enrich their knowledge by reading. It is also sometimes essentials for ordinary people who want to read online newspaper, articles and journals. Therefore, research on text to speech will not only help these visually impaired people but also help mass people.

References

1. Khan M., Alam F., Nath P. – “Bangla Text to Speech using Festival” – CRBLP, Department of Computer Science and Engineering, BRAC University(2011).

2. DR. Khan M., Alam F., Habib M. – “Text Normalization System for Bangla” – CRBLP, Department of Computer Science and Engineering, BRAC University (2011).

3. Schroeter j. – “Text-to-Speech(TTS) Synthesis” – Seminar held in ERB 203, T&T Labs – Research(2011).

4. AlanW Black., Kevin A. Lenzo. – “Building Synthetic Voices” – Language Technologies Institute, Carnegie Mellon University (2010).

5. Rashid, M. M., Hussain, M. A., & Rahman, M. S. (2009, December). Diphone preparation for Bangla text to speech synthesis. In *Computers and Information Technology, 2009. ICCIT'09. 12th International Conference on* (pp. 226-230). IEEE.

6. Mandal, S. D., Warsi, A. H., Basu, T., Hirose, K., & Fujisaki, H. (2010). Analysis and Synthesis of F0 contours for Bangla readout speech. *Proc. of Oriental COCOSDA 2010*.

7. Sen, A. (2004). Bangla Pronunciation Rules and a Text-to-Speech System. *simple’04*.

8. Hoque, M. M., Rahman, M. J., & KumarDhar, P. (2007, March). Lexical Semantics: A Newapproach to Analyze the Bangla Sentence with Semantic Features. In *Information and Communication Technology, 2007. ICICT'07. International Conference on* (pp. 87-91). IEEE.

9. Moulines, E., & Charpentier, F. (1990). Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech communication*, *9*(5), 453-467.

10. Hunt, A. J., & Black, A. W. (1996, May). Unit selection in a concatenative speech synthesis system using a large speech database. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on* (Vol. 1, pp. 373-376). IEEE.

11. Martin, J. H., & Jurafsky, D. (2000). Speech and language processing. *International Edition*.

12. Basu, J., Basu, T., Mitra, M., & Mandal, S. (2009, August). Grapheme to Phoneme (G2P) conversion for Bangla. In *Speech Database and Assessments, 2009 Oriental COCOSDA International Conference on* (pp. 66-71). IEEE.

13. Black, A. W., Lenzo, K., & Pagel, V. (1998). Issues in building general letter to sound rules.

14. Lafferty, J., McCallum, A., & Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

APPENDICES

Bangla Alphabet Soro Barna

অ আ ই ঈ উ ঊ ঋ এ ঐও ঔ

Bangla Banjan Barna

কখ গঘ ঙ চ ছ জ ঝ ঞট ঠ ড ঢ ণ ত থ দ ধ নপ ফব ভ ম য র ল শ
ষহ ড় ঢ় য় □ ং □□ □□

Kar Symbol

□ □◌ ◌□ ◌□ ◌□ ◌□ □□◌ □ □□◌□ □□◌□

Bangla Numeric

১ ২ ৩ ৪ ৫ ৬ ৭ ৮ ৯ ১০