কথা

THE FIRST

TEXT TO SPEECH SYNTHESIS

FOR BANGLA LANGUAGE

A Thesis
Submitted to the Department of Computer Science of
BRAC University
By

Firoj Alam
Student ID: 01201056

Requirements for the Degree of
Bachelor of Science in Computer Science
Spring 2006

BRAC University, Dhaka, Bangladesh

DECLARATION

I hereby declare that this thesis is based on the results found by myself. Materials of work found by other researcher are mentioned by reference. This Thesis, neither in whole nor in part, has been previously submitted for any degree.

Signature of	Signature of
Supervisor	Author
----------	----------
Dr. Mumit Khan	Firoj Alam

ACKNOWLEDGEMENT

First of all, I would like to thank my supervisor, Dr. Mumit Khan. He gave me freedom to choose the thesis topic and complete guidance throughout its development. I was lucky enough to work under him for my thesis work. Although being extremely preoccupied with his busy schedule, he often showed much enthusiasm and took time and lot of pain to review my thesis work that enabled me to improve the system as well as adding new features. I learned plenty of useful things from his comments, revisions and discussions during this period.

I want to give my heartiest gratitude to Roger Tuker, Director of LLSTI (Local Language Speech Technology Initiative) Organization, Ksenia Shalonova, Technical Co-coordinator of LLSTI Organization, and Rob Clark, Lecturer of CSTR (Center for Speech Technical Research) University of Edinburgh.

I want to give thanks to all the faculty members of BRAC University for their helping hands. Thanks to Ali Al Asadullah Mohammad Shafi who supported me in LAB at all time. Also thanks to Hasnat for being with me and always encouraging me.

Thanks to Naira Khan, Lecturer of English and Linguistic of CRBLP (Center for Research of Bangla Language Processing) were always helpful to recognize the typical term of phonetics.

I am also grateful to Naushad UzZaman, Research Programmer of CRBLP, and Zahurul Islam Research Programmer of CRBLP for their support all the time.

Last but not at least, thanks to the Almighty for helping me in every steps of this thesis work.

To the peoples those who dedicated their life for Bangla Language in 1952, those who are working for Bangla Language, as well as my family, friends & well-wishers

Abstract

In this paper, we present Text to Speech (TTS) synthesis system for Bangla language. Here the system developed using phonology, G2P conversion and prosodic information in the festival [1] framework. Since Festival does not provide complete language processing support specific to various languages, so 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 models can be develop and integrate within Festival to develop Bangla TTS system.

TABLE OF CONTENTS

LIST OF TABLE

Chapter I: Introduction

Speech is one of the most vital forms of communication in our everyday life. Since speech is a primary mode of communication among human beings, it is natural for people to expect to be able to carry out spoken dialogue with computers. This involves the integration of speech technology and language technology. Speech synthesis 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. But for Bangla language in Bangladesh there is no such system. The necessary of human computer interactions, a Text To Speech (TTS) system for Bangla language can be overcome the human computer interaction, help to overcome the literacy barrier of common mass, can also empower the visually impaired population and increase the possibilities of improved man-machine interaction through on-line newspaper reading from Internet and enhancing other information system.

Bangla, also known as Bengali, is the language of approximately 210 million people, the majority of whom live in Bangladesh and in the Indian state of West Bengal, making it the 4th most widely spoken language in the world. Since Bangla is the 4th most widely spoken language, it makes greater importance to develop a TTS system for Bangla language.

The function of Text-To-Speech (TTS) system is to convert the given text to a spoken waveform. The TTS system is based on the concatenation of basic speech units of diphones using Festival. The input text is transform into its spoken equivalent by a series of modules. Here we present how these modules constitute the TTS system in detail. The Block Diagram of TTS System is given in figure 1. The input text is essentially a string of characters, might be data from a word processor, i.e text as standard **Unicode** format. The first task is to analyze the raw text. The text may contain number, date, abbreviations, which needs to convert as normal text, which is called text normalization or text analysis. In **chapter II**, we will describe the **Text Analysis**. The normalize text then convert into phonetic representation, which is called phonetic analysis, this can be done by Grapheme to Phoneme (G2P) conversion or dictionary lookup or combination of both. In **chapter III**, we present the **Phonetic Analysis**. After

phonetic analysis we need to add stress and intonation in sentences, words and syllable as needed, which is called prosodic conversion. In **chapter IV**, we propose some issues about the **Prosodic Conversion**. In chapter V, we present **Waveform Synthesis**, which is done by multisyn unit selection technique. In chapter VI, introduces about integration with BanglaPad then in chapter VII, we introduce future plan, after that conclusion.

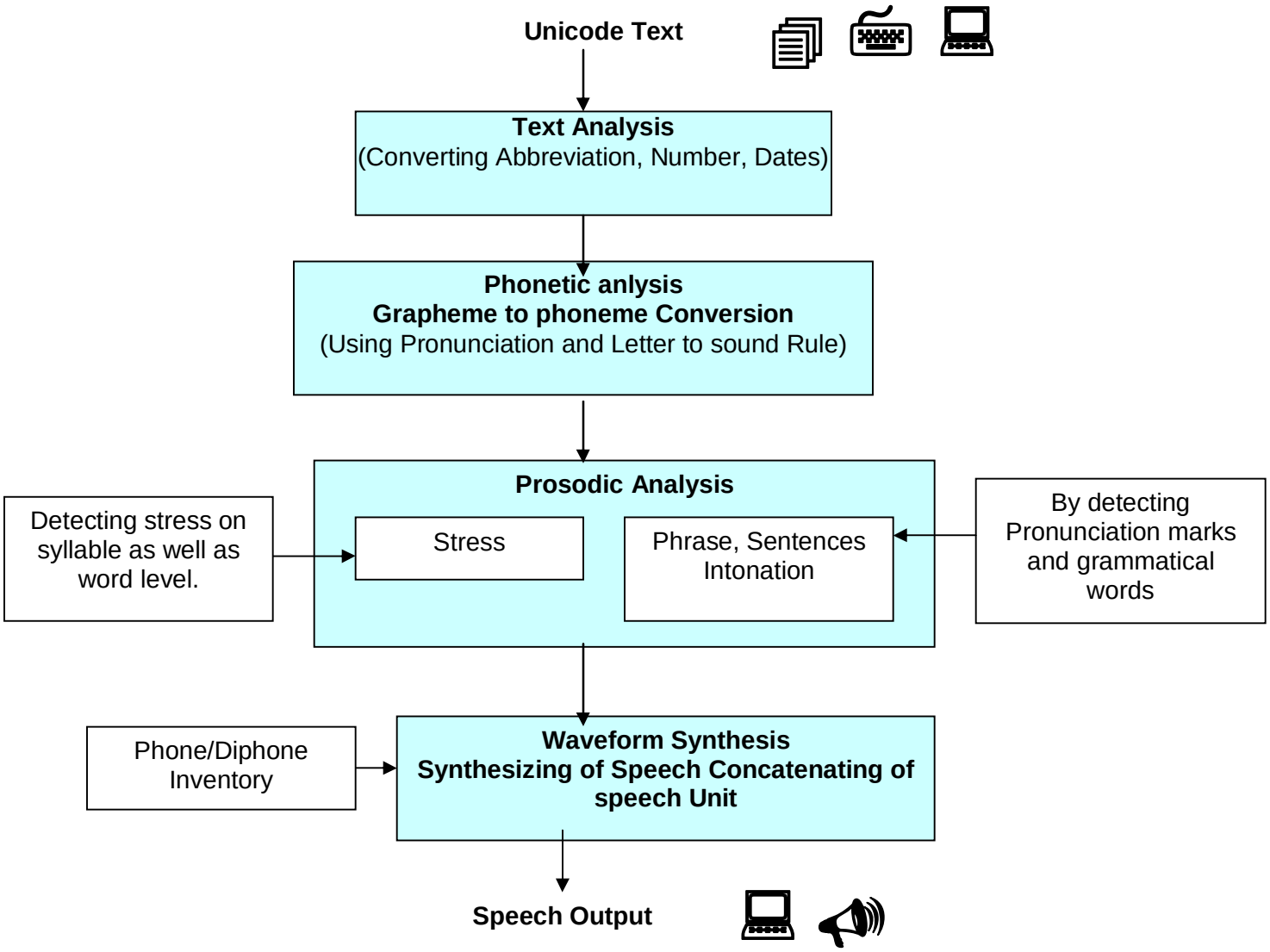

Fig 1: Bangla Text To Speech Block Diagram

Chapter II: Text Analysis

2.1 Definitions:

The first step of Text To Speech system is text analysis that means analysis of raw text into pronounceable words. It involves the work on the real text, where many Non-Standard Word (NSW) [2] representations appear, for e.g., numbers (year, time, ordinal, cardinal, floating point), abbreviations, acronyms, currency, dates, URLs. All of these non-standard representations should normalize, or in other words convert to standard words. These NSW should normalize using text normalization and ambiguous token should disambiguate using homograph disambiguation.

2.2 Text normalization:

Text normalization typically involves tokenization of input text into tokens, identification of Non Standard Words (NSWs) and their categories, and expansion of the NSWs into standard word representations. We can tokenize our text based on white spaces [Sproat et al., 2001]. Once tokenization is done, each token has to identify for its corresponding category. After identifying the category, expansion of NSWs can accomplish by a combination of some steps (e.g., for expanding numbers, currency, dates) and look-up tables (e.g., for abbreviations, acronyms). Work on text normalization in TTS systems mostly involves a set of rules. As our Bangla text contains some of the issues such as: Text may contain intersperse English words using Roman script, making it a bilingual text. Numbers represent by English numerals at one place, and the native Bangla numerals at another place, in the same text.

2.3. Homograph Disambiguation:

A "homograph" is a word with the same text as another word, but with a different pronunciation. Identification of token category involves a high degree of ambiguity, for example, '1974' could refer to nineteen seventy four as a year, or one thousand nine hundred seventy four as a cardinal number. Disambiguations generally handle manually using some rules. However, such rules are very difficult to write, maintain, and adapt to new domains. Bangla language does not have much more homograph ambiguity except digits and years. Like, the digits "1997" might be spoken as "nineteen ninety-seven" if someone is talking about the year, or "one thousand nine hundred and ninety seven" if someone is talking about the number.

2.4 How to proceeds using festival:

2.4.1 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 [3]. These phone 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 chart [4] is given in the next consecutive table.

Consonants/e¨vÄbeY

Manner of articulation		**Place of articulation**										
		Velar		**Palato-alveolar**		**Alveolar**		**Dental**		**Bilabial**		**Glottal**
		In-aspirate – **AÍcvY**	Aspirate- **gvNcvY**	In-aspirate – **AÍcvY**	Aspirate- **gvNcvY**	In-aspirate - **AÍcvY**	Aspirate- **gvNcvY**	In-aspirate - **AÍcvY**	Aspirate- **gvNcvY**	In-aspirate – **AÍcvY**	Aspirate- **gvNcvY**	
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	**O 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 /	**Aw/** a /	**B-C/** i /	**D-E/** u /	**F/** ri /
G/e /	**H/** oi /	**I/** o /	**J/** ou /	

Vowel		**A/**O /	**Aw/** a /	**B-C/** I /	**D-E/** u /	**F/** ri /	**G/**e /	**H/** oi /	**I/** o /	**J/** ou /	**G¨w/** { /
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 : Bangla transliteration table on vowel

Modifiers **

Symbol with [k□] (□(	Name	Function	Transliteration
□□	hôshonto	Suppresses the inherent vowel	-
□□□	khôndđo 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

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

Table 5 : Bangla transliteration table on Consonants.

Digits

Arabic numerals	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Bangla numerals	□	□	□	□	□	□	□	□	□	□
Bangla names	shunno	ek	dui	tin	char	paac	chy	shaat	aat	ny
	□□□□□	□□	□□□	□□□	□□□	□□□□	□□□	□□□	□□	□□□

Table 6 : Bangla transliteration table on Digits

2.4.2. Transliteration by scheme:

We can skip first step by mapping Unicode text to roman script in scheme. This system is done by Telugu language.

2.4.3. Text Normalization:

We propose a better approach to text normalization [5] within festival, wherein tokenization and initial token classification can combine into one stage, which can use by a second level of token sense disambiguation. Tokenization and initial token classification can perform using a scheme programming that can derive from various token definitions in the form of regular expressions.

Steps in Text Normalization:

1) Split the token (based on whitespace and punctuation)
2) Type identifier: for each split token identify type.
3) Token expander: for each typed token, expand to words
    - Deterministic for number, date, money, letter sequence.
    - Nondeterministic for abbreviations.

2.4.3.1. Split the token:

We can split our token based on whitespace and punctuation.

Here

- Whitespace can be viewed as separators.
- Punctuation can separate from the raw tokens.
- Festival converts text into
  - ordered list of tokens
  - each with features:
    - its own preceding whitespace
    - its own succeeding punctuation

Whitespace is the most commonly used delimiter between words and is extensively used for tokenization. But using whitespace as the only delimiter have some limitation: a token type which allows the occurrence of whitespace within the token will not recognize as a single token, but split up into two or more tokens. For example, consider a telephone number 880 2 9567447. This can identify as a single token of type ‘Telephone Number', but if tokenization is exclusively based on whitespace, then we end up having 3 tokens. Further, an important limitation is that every token will then have to go through a token identification process that identifies its token type/category.

One of issue in tokenization is end of utterance detection. It is done by detecting [. for dhari, ? question marks, ! for exclamation]. Festival use decision tree in the backend, we just provide punctuation marks as front-end input.

There is also an ambiguity in our abbreviations. We use colon [**:**] for abbreviation as well as middle of two sentences.

E.g:

wkí cwZôvb eÜ: jvMvZvi niZvj I 144 avivi Kvi‡b Kvbmv‡Ui wkí cwZôvb¸‡jv eÜ i‡q‡Q|

W: [W±i]

Av: [Avãj]

For the second level of token sense disambiguation (i.e number and year), we can use decision lists and decision trees. Until now we didn’t concentrate on decision list and decision tree.

2.4.3.2. Token Identification:

Using scheme programming and regular expression we can classify (split the token) and identify the token of various NSWs [6]. Different formats of each Non Standard Word (NSW) category are depend on the regular expressions. We can address the following non-standard representations:

- Cardinal numbers and Literal Strings E.g 1,2,3 etc.
- Ordinal numbers. E.g: 1st, 2nd, 3rd, etc.
- Roman Numerals. E.g: I, II, III, etc.
- Fractions. E.g: ½, ¾, etc.
- Ratios. E.g: 1:2, 3:4
- Decimal Numbers. E.g: 100,1000
- Alphanumeric strings
- Time. E.g: 12.12, 12:12
- Telephone Numbers. E.g: 880 2 9567447, 880-2-9567447
- Date. E.g: 12/2/06, 12-2-06.
- Year. E.g: 2006, 1997
- URL and E-mail. E.g: firojalam04@yahoo.com
- Range. E.g: 123-1000
- Percentage. E.g: 12%

In this stage the scheme regular expression is run on each sentence, it analyses the text looking for strings, which match one of its patterns. So, by defining regular expressions that match the formats of the various token types, we can automatically extract the token that best fits the given token description. In case of ambiguity between two or more token categories/types for a particular token, the lexical analyzer should configure to output the possible categories with the token to facilitate token sense disambiguation at a later stage. Using this approach, we can complete tokenization and initial token classification (most of the NSW category identification).

As the tokenization and initial token classification is done, then we can easily identify NSW token category. After that we can use this information (the complete meaningful token name) in the phonetic analysis modules of a TTS system, which can help in appropriate rendering of the speech.

2.4.3.3. Expanding NSW Tokens:

At this stage we have all NSW tokens. Now we have to expand it to full text. Suppose a telephone number 880 2 0152303398 is tokenized and identified as a single meaningful token (NSW) type “Telephone Number” (instead of tokenizing the telephone number into 3 tokens and a further token identification process for each of the tokens), the complete meaningful token name can use effectively in the phonetic analysis modules of a TTS system. For example expanding a token is done by the following way:

Number expands to string of words representing cardinal.

Year 1995 expand to 2 pairs of number digits. E.g: Year: □□□□- **Dwbk kZ** evqvb□

Telephone number 880 2 0152303398 expand to string of digits.

Phone: □□□□□□□- □□ □□□□ □□ □□□ □□ □□ □□□

Abbreviation expands to itself. □: - □□□□□

And so on.

2.4.4. Homograph Disambiguation in festival:

Though Bangla text does not have much more ambiguity, but there is an ambiguity in number/year and time/float. To solve this we can use decision list and decision tree. For decision list and decision tree building process, “Wagon” classification and “regression tree” tool [SpeechTools, 2002] can use. **This is still research issue**.

Chapter III: Phonetic Analysis

In this chapter we present the method for finding the pronunciation of a word. This is either by a lexicon (a large list of words and their pronunciations) or by some method of letter to sound rules. We can use large list of lexicon. In our implementation I used 900 lexicons.

3.1. Definitions:

In a phonological orthography a **grapheme** corresponds to **phoneme**. That is how to pronounce a written text.

The root of Bangla language is Sanskrit, which is phonetically perfect (i.e. there is very little or almost no discrepancy between written text and pronunciation), Bangla is pronounced almost as it is written. There are 48 phonemes in Bangla language, but there are 5 phonemes (anusker, bisurga, chandrabindu, nio, umma) that are pronounced with the help of other phoneme. They are not pronounced individually. For G2P conversion we have done one-to-one mapping for text to phone. At the first step Bangla Unicode letter to English letter does transliteration. The transliteration is given in table 1.

3.2. Steps of Phonetic Analysis:

1. Building large amount of lexicon by hand.
2. Building letter-to-sound rules by hand.

3.2.1. Building large amount of lexicon by hand:

Initially I added lexicon then I added letter to sound rule. As there is no difference between Bangla written text and pronunciation. Building a letter to sound rule is much easier by hand, though letter to sound rule can be done automatically for large set of corpus by training. But there is certain exception about (anusker, bisurga, chandrabindu, nio, umma), conjugate cluster; we have to deal it by rule in future. Our letter to sound rules incomplete yet.

A pronunciation in Festival requires not just a list of phones but also a syllabic structure. We already discussed about our phones with their features. 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 our system I developed large set of lexicon based on our syllabic structure. An example entry of lexicon is

("aapni" n (((aa p) 0) ((n i) 0)))

Explicit marking of syllables a stress value is also given (0 or 1). Rules of syllabification [7] are given here:

c - Consonant, v-vowel

B (i), q (j), e (w)

1. **v** **G, I**
2. **vc** **AvR, Avg, Gi**
3. **cv** **cv, `v, gv**
4. **cvc** **KvR, bvK, †PvL**
5. ccv **Kwl, `p**
6. cccv ¯x
7. ccvc **cvY, Îvb**
8. vi **GB, IB**
9. cvi **w`B, wkDwj, mB**
10. **vj** **Avq**
11. **cvj** **b¨vq, Ab¨vq**
12. ccvj **cvq**
13. vw **Jla** **(GwU ¯Zšfv‡e cY kã MVb K‡i bv)**
14. vwc **JrmK** **(GwU ¯Zšfv‡e cY kã MVb K‡i bv)**
15. **cvw** **`vl, bvl**
16. cvcj **b‡q,**
17. *wv **lqwik**
18. *yv **‡g‡q**
19. wvw **Lvlqvl**
20. wvj **‡blqvq**
21. jvc **c‡qvM**

The most common rules v, vc, cv, cvc, vj, cvj are used for syllabification.

The basic assumption in Festival is that we will have a large lexicon, tens of thousands of entries, that is a 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. 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 in letter-to-sound rules. In our implementation I included 900 lexicons from a text corpus. This lexicon can add manually or automatically. Here we added it manually by hand. Explicitly we used our syllabification rule.

3.2.2. Building letter-to-sound (LTS) rules by hand:

In Festival there is a letter to sound rule or grapheme to phoneme (G2P) system that allows rules to be written, but Festival also provided a method for building rule sets automatically, which will often be more useful. There is no difference between orthography and its phone set of Bangla Language, so latter-to-sound (LTS) rule can be written by hand. We used LTS rule in our implementation, but it is still a **research issue**. We implemented this LTS rule based on our syllabification rule.

3.3. Out of vocabulary words:

It is impossible to list all words in a natural language for general text-to-speech. We need to provide something to pronounce out of vocabulary words i.e large number of names, local language. By default a lexicon in Festival will throw an error if a requested word isn't found. Festival uses *symbolexplode* function. This recursively calls the lexical lookup function on the characters in a word. Each letter should appear in the lexicon with its pronunciation (in isolation). The *symbolexplode* function assumes that letters are single bytes, which may not be true for some cases and that function would need to be replaced for that cases. Note that we append the syllables of each of the letters in the word. For long words this might be too naive as there could be internal prosodic structure in such a spelling that this method would not allow for. In that case festival builds letters to be words thus the symbol explosion to happen at the token to word level. This *symbolexplode* function may be the worst solution. We have to build LTS rule or G2P rule for proper noun (names) and out of vocabulary words. To handle the out of vocabulary word is also a **research issue**.

Chapter IV: Prosodic Analysis

4.1. Phrasing Definitions:

Prosodic phrasing in speech synthesis makes the whole speech more understandable. Due to the size of people lungs there is a finite length of time people can talk before they can take a breath, which defines an upper bound on prosodic phrases. However we rarely make our phrases this maximum length and use phrasing to mark groups within the speech. There will be a substantial number of prosodic boundaries, which are not explicitly marked with punctuation. Thus a prosodic phrasing algorithm solely based on punctuation will typically under predict but rarely make a false insertion. However depending on the synthesizer it may be the case that explicitly adding punctuation at desired phrase breaks is possible and a prediction system based solely on punctuation is adequate.

4.2. Phrasing Methods:

Festival supports two major methods, one by rule and the other using a statistical model based on part of speech and phrase break context.

4.2.1. Phrasing by decision tree:

The rule system uses a decision tree, which may be trained from data or hand written. The first basic method of festival is CART tree [8]. 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 you consider is appropriate form break labels as long as the rest of your 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. These simple punctuations are used in our implementation to predict phrase break or pause insertion.

```
(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 model is often reasonable for simple TTS, but for longer context with no punctuation it does not work well. For a complete TTS lot of information is required to build CART tree.

4.2.2. Phrasing by statistical models:

The second method of phrasing is statistical models. Most methods try to find the likelihood of a break after each word but don't take into the account the other breaks that have already been predicted. This can often predict breaks at places where there is a much more reasonably place close by. The more elaborate model supported by Festival finds the optimal breaks in an utterance, based on the probability of a break after each word (based on its part of speech context), and the probability of a break based on what the previous breaks are. This method is not implemented in our system until now, but for better phrasing model we have to do it in future.

However it should be noted that without a good intonation and duration model, spending time on producing good phrasing is probably not worth it. The quality of all these three prosodic components (phrasing, intonation, and duration) is closely related such that if one is much better than others, it may not be any real benefit.

4.3. Intonation:

There are probably more theories of intonation that 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:

1. **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).
2. **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.

4.3.1. Accents/Tone Assignment:

Accent is a property of a word in context - it is a way to mark intonational 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** [9] (for building CART tree) can be used to predict accents, and similar tree should be used to predict boundary tones as well.

4.3.2. 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, I used this simple method for F0. For larger context we have to think about ToBI (Tones and Break Indices) is an intonational 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 I used these fixed value. This is taken from Kiswahili TTS system [10].

```
(set! bu_bangla_bappi::phone_durs
'(
  ;;; PHONE DATA
  ;; name zero mean in seconds e.g.
  ; all phones on bu_bangla phoneset
   (# 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)
```

(u 0.0 0.100)
(uu 0.0 0.110)
(e 0.0 0.110)
(oi 0.0 0.110)
(o 0.0 0.110)
(ou 0.0 0.110)
……
……...

4.5. Intonation pattern for Bangla:

We have some intonation rule in Bangla language [11]. 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: articulatory 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 multisyn unit selection technique [12] for waveform synthesis (that means to build our voice database). Recording is one of the big issues in these techniques. So professional speaker should record our voice in a clean environment. In this chapter we present multisyn voice building process that we followed step by step.

5.1. How to Builds new voice using Limited domain technique:

This technique also uses unit selection methodology. To build a new voice it is easier than multisyn unit selection technique. It will take 2/3 days to build a new voice if festival is configured properly in Linux PC. But we spent more time to build our voice because of our configuration took longer time.

5.1.1. How to install tools:

We presume our operating system is Fedora Core 4. If it is later version then may be we will face version problem. To solve this problem, we have contact festvox mailing list.

Install the latest version of

Festival-1.9.6,

Speech_tools-1.2.9.6,

Festvox-2.0

The latest version of festival and speech_tools are available in the following link.

http://www.speech.cs.cmu.edu/awb/fftest/speech_tools-1.2.96-beta.tar.gz

http://www.speech.cs.cmu.edu/awb/fftest/festival-1.96-beta.tar.gz

Other versions are available in the following link.

http://festvox.org/packed/festival/1.95/

We should install voices, that is available in the above link
festvox_kallpc16k.tar.gz
festlex_POSLEX.tar.gz
festlex_CMU.tar.gz
festvox_cmu_us_slt_arctic_hts.tar.gz
festvox_cmu_us_jmk_arctic_hts.tar.gz
festvox_cmu_us_bdl_arctic_hts.tar.gz
festvox_cmu_us_awb_arctic_hts.tar.gz

Extract festival, speech_tools, and festvox to your home directory.
Extract all by using the command: $ tar –zxvf [file name]

$tar -zxvf speech_tools-1.2.96-beta.tar.gz

We will get the festvox 2.0 in the above link http://www.festvox.org/
Now we can follow the steps:
$cd speech_tools
$make
$cd ../festival
$make
If every thing works properly then we have to test festival, either is it works properly or not.
$festival/bin/festival
Or
$ cd festival/bin
$festival
festival>(SayText “Hello World”).

We presume here the system will produce sound.

More information is available in festival document.

5.1.2. How to Build:

Here we assume festival works perfectly.
Extract festvox by
$tar –zxvf festvox-2.0-release.tar
$cd festvox
$make

Here we have to setup your environment variable by the following command:
export FESTVOXDIR=$HOME/festvox
export ESTDIR=$HOME/speech_tools
export FESTIVAL=$HOME/festival/bin/festival
export LD_LIBRARY_PATH=$HOME/speech_tools/lib:$LD_LIBRARY_PATH

$mkdir demo
$cd demo
$FESTVOXDIR/src/ldom/setup_ldom bu bangla firoj
#bu-organization name, bangle- language name, firoj- speaker name.

As in the definition of diphone databases we require three identifiers for the voice. These are institution, domain and speaker. The primary reason for these is that people do not all build limited domain synthesizer with the same thus making it not possible to load them into the same instance of festival. This is just a convention.
After executing the above command we will get the whole branch of directory.
Make utts.data file with the following format and keep it in demo/etc/ directory:

(sen001 "aapni hytaobaa kmpiutaar er saamne bse aachen. ")
(sen002 "kichu kraar paacchen naa. ")
(sen003 "haatae kichu smy aache ki ")
……
……

The format is "(" followed by a filename, root followed by the text for that sentence, followed by ")" each on separate lines. This text when converted to a phone

sequence by Festival should match (as closely as possible) the phone sequence of the speech. With this in mind we should probably ensure all words are in your lexicon (if we are using one) and it is probably best to write numbers and dates out in full as they were spoken. (e.g "the ninth of May" rather than "9 May" etc...)

$FESTIVAL -b festvox/build_ldom.scm '(build_prompts "etc/demo.data")'

The next command (prompt_them) is used for voice recording. We can record our voice in linux system by the following command. Advantage is that we do not have to label of our voice. But we have to careful about our voice recording. Recording voice will be automatically saved in /home/demo/wav directory. After recording of our voice, we have to test that, our recording voice and utts.data sentences matches perfectly.

$bin/prompt_them etc/utts.data
Execute the following command sequentially.
$bin/make_labs prompt-wav/*.wav
$FESTIVAL -b festvox/build_ldom.scm '(build_utts "etc/demo.data")'
$bin/make_pm_wave wav/*.wav
$bin/make_pm_fix pm/*.pm
$bin/simple_powernormalize wav/*.wav
$bin/make_mcep wav/*.wav
$FESTIVAL -b festvox/build_ldom.scm '(build_clunits "etc/demo.data")'
$FESTIVAL festvox/bu_bangla_firoj_ldom.scm '(voice_bu_bangla_firoj_ldom)'

festival> (SayText "aapni hytaobaa kmpiutaar er saamne bse aachen.")

5.1.3. Limitations:

1. In larger context it is not better.
2. We can’t add our lexicon, phonesets, and LTS rule.
3. If segment mismatch occur it generate speech by its default voice.

5.2. Building voice Using Multisyn Unit Selection technique:

Building new voice using this technique is something critical. This is real application where we can put our phonesets, lexicon, and voices. In this section we propose how to install tools and how to build voice.

5.2.1. How to install tools:

This is the same as section 5.1.1. Also we have to add other tools, that's are

multisyn_build latest version,

python [for generating mfcc].

Swig

Perl

HTK

Multisyn build is available at http://www.cstr.ed.ac.uk/downloads/festival/multisyn_build

Python, Swig are included with Fedora core 4. We have to configure these tools.

HTK is available in the following link: http://htk.eng.cam.ac.uk/download.shtml

Extract and make multisyn_build and HTK.

To configure python, swig, and perl we have to change the following files.

In speech_tools/makefile

Comment out "wrappers" that is available in the following line.

EXTRA_DIRS=siod java rxp #wrappers

In speech_tools/config/config file:

set this line:

SHARED = 2

Comment out these line

CONFIG_SWIG_COMPILER = /usr/bin/swig

CONFIG_WRAPPER_LANGUAGES = PYTHON PERL5

We have to setup our appropriate version that is available in fedora core 4.

Language specific includes should be set to correct site paths

#CONFIG_PYTHON_INCLUDES= -I/usr/include/python2.2/

CONFIG_PYTHON_INCLUDES= -I/usr/include/python2.4/

#CONFIG_PERL_INCLUDES= -I/usr/lib/perl5/5.8.3/i386-linux-thread-multi/CORE/
CONFIG_PERL_INCLUDES= -I/usr/lib/perl5/5.8.6/i386-linux-thread-multi/CORE/

Then we have to remake

1. speech_tools, and then
2. festival

All configuration information is available in multisyn_build/doc/notes.
Install HTK-3.3. This is necessary for labeling by alignment. We didn't follow this procedure for labeling. We followed different procedure, which describes next.

5.2.2. How to Build:

Here we can assume that our entire configuration is perfect. We can start voice-building process.

Setup environment variable by the following command:

export FESTVOXDIR=$HOME/festvox
export ESTDIR=$HOME/speech_tools
export HTK=$HOME/htk-3.3/bin.linux
export FESTIVAL=$HOME/festival/bin/festival
export LD_LIBRARY_PATH=$HOME/speech_tools/lib:$LD_LIBRARY_PATH

Build unit selection voice by festvox

#festvox Unit selection
mkdir bu_bangla_bappi
cd bu_bangla_ bappi
$FESTVOXDIR/src/unitsel/setup_clunits bu bangla bappi
#bu-organization name, bangle- language name, bappi- speaker name.
After executing the above command we will get the whole branch of directory.
We have to put of our phoneset in festvox/bu_bangla_bappi_phoneset.scm file
Add all lexicons in /home/bu_bangla_ bappi /festvox/bu_bangla_bappi_lexicon.scm file. How to add lexicon discussed in section 3.2.1. Then make utts.data file with the following format:
(sen001 "aapni hytaobaa kmpiutaar er saamne bse aachen. ")
(sen002 "kichu kraar paacchen naa. ")

(sen003 "haatae kichu smy aache ki ")
……

The format is "(" followed by a filename, root followed by the text for that sentence, followed by ")" each on separate lines. This text when converted to a phone sequence by Festival should match (as closely as possible) the phone sequence of the speech. With this in mind we should probably ensure all words are in our lexicon (if we are using one) and it is probably best to write numbers and dates out in full as they were spoken. (e.g "the ninth of May" rather than "9 May" etc...)

We have to keep utts.data file in /home/bu_bangla_bappi/etc directory

#festvox command for .lab
$FESTIVAL -b festvox/build_clunits.scm '(build_prompts "etc/utts.data")'

The next command (prompt_them) is used for voice recording. We can record our voice in linux system by the following command. Advantage is that we do not have to label of our voice. But we have to careful about our voice recording. Recording voice will be automatically saved in /home/bu_bangla_bappi/wav directory. After recording our voice, we have to test that, our recording voice and utts.data sentences matches perfectly.

$./bin/prompt_them etc/utts.data
$./bin/make_labs prompt-wav/*.wav

After that we will get the .lab files in the /home/bu_bangla_bappi/lab directory. This lab files will be input of our multisyn voice. Then we have to follow the next following steps. We have to put .lab files in /home/bu_bangla_hasnat/lab after creating bu_bangla_hasnat.

$ mkdir /home/bu_bangla_hasnat (or any arbitrary directory name)
$ cd /home/ bu_bangla_hasnat
$../multisyn_build/multisyn_build.sh
$../multisyn_build/bin/setup

We have to put wav files **from** /home/bu_bangla_bappi/wav **to** /home/bu_bangla_hasnat/wav
We have to put festvox directory **from** /home/bu_bangla_bappi/ **to** /home/bu_bangla_hasnat/

We should power normalizes our wav file to find better frequency. The script file "simple_powernormalize.sh" is available in demo/bin. We have to keep it multisyn_build/bin/
Then we have to use this command:
$../multisyn_build/bin/simple_powernormalize wav/*.wav
All normalized file will be available in bu_bangla_hasnat/wavn
So we have to put all *.wav files **from** bu_bangla_hasnat/wavn **to** bu_bangla_hasnat/wav

$ mkdir pm
$ ../multisyn_build/bin/make_pm_wave -m pm wav/*.wav
(or -f to select default female parameters)
$ ../multisyn_build/bin/make_pm_fix pm/*.pm

Utterance building:
The output of the program are the .utt and .lab files generated in the folder /home/bu_bangla_hasnat/utt/*.utt and *.lab

One of the most difficult procedures in voice compilation is the utterance building. The section "Generate utterances" is available in the file multisyn_build/notes.

The following files require as input for utterance building:
-scm files (located in bu_bangla_hasnat/fextvox/*.scm)
-lab files (generated from unit selection technique)
-pm files
Pitchmark (pm/*.pm files) generated from the following command

../multisyn_build/bin/make_pm_wave -m pm wav/*.wav
../multisyn_build/bin/make_pm_fix pm/*.pm)

-utts.data (file with the orthographic representation of .lab files)
-build_unitsel.scm (the file contains paths to bangla scheme files that are stored in festvox directory). The file build_unitsel.scm can remain in multisyn directory and can be called from there.

We have to edit /multisyn_build/scm/build_unitsel.scm and include the following lines in build_utts method:

```
    ((string-equal lexicon '' bu_bangla '')
    ;; Explicitly select phoneset.
    (require 'festvox/ bu_bangla _hasnat_phoneset)
    (bu_bangla _hasnat::select_phoneset)
  ;; And lexicon.
    (require 'festvox/ bu_bangla _hasnat_lexicon)
    (lex.select lexicon)
    ;; And durations
    (require 'festvox/ bu_bangla _hasnat _durdata)
    (Parameter.set 'Duration_Method 'nil)
    (require 'festvox/ bu_bangla _hasnat _tokenizer)
    (bu_bangla _hasnat::select_tokenizer)
    ;; And post lexical rules
    (set! postlex_rules_hooks nil)
    ;; Use unilex Word module.
    (Parameter.set 'Word_Method Classic_Word))
```

$mkdir utt (in /home/ bu_bangla _hasnat)
$FESTIVAL ../multisyn_build/scm/build_unitsel.scm
festival> (build_utts "utts.data" ' bu_bangla)
festival> (exit)

$mkdir f0
$../multisyn_build/bin/make_f0 -m wav/*.wav
$mkdir lpc
$$FESTVOXDIR/src/general/make_lpc wav/*.wav

Now we have to generate mfcc file from our wav file.
$mkdir aligment
$../multisyn_build/bin/make_mfccs alignment wav/*.wav
This mfcc files will be used to generate coefs: we have to use the following command to generate coefs and stripped coef
$../multisyn_build/bin/make_norm_join_cost_coefs coef f0 mfcc/ '.*\.mfcc'
$../multisyn_build/bin/strip_join_cost_coefs coef coef_stripped utt/*.utt

Generate directory structure:
mkdir /home/bangla
mkdir /home/bangla/bu_bangla_hasnat_multisyn
mkdir /home/bangla/bu_bangla_hasnat_multisyn/db
mkdir /home/bangla/bu_bangla_hasnat_multisyn/files_scm
mkdir /home/bangla/bu_bangla_hasnat_multisyn/db/coef_stripped
mkdir /home/bangla/bu_bangla_hasnat_multisyn/db/lpc
mkdir /home/bangla/bu_bangla_hasnat_multisyn/db/utt

We have to copy pauses directory and utt.pauses from **Kswahili_voice** then we have to keep it in /home/bangla/bu_bangla_hasnat_multisyn/db/
Copy utts.data **from** /home/bu_bangla_ bappi/ **to** /home/bangla/bu_bangla_hasnat_multisyn/db/
Copy *.scm files **from** /home/bu_bangla_bappi/festvox **to** /home/bangla/bu_bangla_hasnat_multisyn/files_scm/
Copy *.lpc and *.res files **from** /home/bu_bangla_bappi/lpc **to** /home/bangla/bu_bangla_hasnat_multisyn/lpc/
Copy *.utt and *.lab files **from** /home/bu_bangla_bappi/utt **to** /home/bangla/bu_bangla_hasnat_multisyn/utt/
Copy your *.coef **from** /home/bu_bangla_bappi/ coef_stripped **to** /home/bangla/bu_bangla_hasnat_multisyn/ coef_stripped/

mkdir /home/festival/lib/voices-multisyn/bangla
mkdir /home/festival/lib/voices-multisyn/bangla/bu_bangla_hasnat_multisyn
mkdir /home/festival/lib/voices-multisyn/bangla/bu_bangla_hasnat_multisyn/festvox/
Copy UON_swahili_kw_multisyn.scm **from** festival/lib/voices-multisyn/kiswahili/UON_swahili_kw_multisyn/festvox **to** /home/festival/lib/voices-multisyn/bangla/bu_bangla_hasnat_multisyn/festvox/ then rename as bu_bangla_bappi_multisyn.scm. We have to edit this file appropriately.
NOTE: We followed the Kswahili example to build directory structure. We can copy pause module from Kswahili for our voice. It will be better than generating.
Kswahili example available at http://www.llsti.org/downloads-languages-kiswahili.htm.

Another way of creating pauses and directory structure is available in multisyn_build/doc/notes.

Chapter VI: Integration with BanglaPad

6.1 BanglaPad:

BanglaPad is an open source, full-featured cross-platform Unicode rich text editor capable of editing Bangla that can run on different operating systems, such as Windows, Linux/Unix, owing to its base on the Java programming language. Users can type Bangla text without using external helper applications, such as keyboard drivers and can check spelling of both Bangla and English document.

6.2 How to integrate with BanglaPad:

We can integrate our TTS system with BanglaPad so that user can write Bangla in BanglaPad and can hear the speech of the corresponding text. This involves some technical issues:

- First of all we have to follow the installation and developing procedure of TTS system that is explained in section 5.
- To integrate with BanglaPad we have add a button within BanglaPad, under that we have to call our Bangla to English script program (Main.java available in our packages) for transliteration.
- Then we have to call the program (ScriptBuilder.java available in our packages) that runs shell script to generate text to speech using festival.

Chapter VII: Future Works/ TODO

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:**
    a. Automatic lexicon entries instead of adding manually.
    b. Find out LTS or G2P rule.
3. **Prosody Analysis:** Have to build CART tree for Bangla intonation pattern using ToBI or wagon.
4. **Waveform synthesis:**
    a. We have to overcome the limitation of multisyn unit selection technique.
    b. Have to build diphone database for better sound quality. This is one of the huge research issues.

Chapter VIII: Conclusion

The described speech synthesis system is the first 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 multisyn unit selection 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. So our system will not only help these visually impaired people but also help mass people.

References

(1) Festival Speech Synthesis, Speech Tools & documentation, http://www.festival.org/

(2) Sproat, R., Black, A., Chen, S., Kumar, S., Ostendorf, M., and Richards, C. 2001. Normalization of Non-standard Words, Computer Speech and Language, 15(3):287-333 http://www.clsp.jhu.edu/ws99/projects/normal/slides/intro/nswintro.pdf

(3) Phoneme set and their features, (1) Naira Khan, (2) Basa Bigganer Kotha by Daniul Haque, (3) Dhani Biggan O Dhanitotto by Abdul Hai

(4) Bengali script – Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Bengali_script

(5) Jurafskey Lectures, Spring 2006, www.stanford.edu/class/cs224s/

(6) Hindi Text Normalization, K. Panchapagesan, Partha Pratim Talukdar, N. Sridhar Krishna, Kalika Bali, A. G. Ramakrishnan, Hewlett-Packard Labs India, 24 Salarpuria Arena, Hosur Road, Bangalore, India. Email: {*partha.talukdar, nsridhar, kalika}@hp.com* Indian Institute of Science Bangalore, India. Email: *ramkiag@ee.iisc.ernet.in* Birla Institute of Technology & Science Pilani, Rajasthan, India Email: *panchapagesan.k@gmail.com*

(7) Rules of syllabification, Dhani Biggan by MD. Abdul Hay, pg-152

(8) CART tree description, Speech tools documentation. http://www.festival.org/

(9) Wagon description, Speech tools documentation. http://www.festival.org/

(10) Kiswahili TTS system, www.llsti.org

(11) Implementation of intonation pattern in bengali text to speech synthesis, an approach, Asok Bandyopadhyay, Shyamal Kr. Das Mandal, Barnali Pal, Mridusmita Mitra Speech& Signal Processing Group ,ER&DCI,Calcutta Plot E2/1,Block GP,Sector V,Saltlake ,Kolkata-700 091

(12) Multisyn Unit selection technique, Rob Clark, http://www.cstr.ed.ac.uk/downloads/festival/multisyn_build

Appendices

A: Bangla Alphabet

Soro Barna

□ আ ই ঈ উ ঊ ঋ □ ঐ
ও ঔ

Banjan Barna

□ খ গ ঘ ঙ চ ছ □ ঝ ঞ
ট ঠ ড ঢ ণ ত থ দ ধ ন
প ফ ব ভ ম য র ল শ ষ
হ ড় ঢ় য় □ ং ঃ ঁ

Kar Symbol

□ ি ী ু ূ ৃ ে □ ো ৌ

Bangla Numeric

□ ২ ৩ ৪ ৫ ৬ ৭ □ ৯ ১০

B: Bangla Unicode Chart

Bengali

Range: 0980–09FF

This file contains an excerpt from the character code tables and list of character names for the Unicode Standard, last updated for
The Unicode Standard, Version 4.0.

This file may be updated as necessary to reflect errata without notice. For an up-to-date list of errata, see *http://www.unicode.org/errata/*

Disclaimer

These charts are provided as the on-line reference to the character contents of the Unicode Standard, Version 4.0 but do not provide all the information needed to fully support individual scripts using the Unicode Standard. For a complete understanding of the use of the characters contained in this excerpt file, please consult the appropriate sections of The Unicode Standard, Version 4.0 (ISBN 0-321-18578-1), as well as Unicode Standard Annexes #9, #11, #14, #15, #24 and #29, the other Unicode Technical Reports and the Unicode Character Database, which are available on-line.

See http://www.unicode.org/Public/UNIDATA/UCD.html and http://www.unicode.org/reports/

A thorough understanding of the information contained in these additional sources is required for a successful implementation.

Fonts

The shapes of the reference glyphs used in these code charts are not prescriptive. Considerable variation is to be expected in actual fonts. The particular fonts used in these charts were provided to the Unicode Consortium by a number of different font designers, who own the rights to the fonts.

See http://www.unicode.org/charts/fonts.html for a list.

Terms of Use

You may freely use these code charts for personal or internal business uses only. You may not incorporate them either wholly or in part into any product or publication, or otherwise distribute them without express written permission from the Unicode Consortium. However, you are welcome to provide links to these charts.

The fonts and font data used in production of these Code Charts may NOT be extracted or otherwise used in any commercial product without permission or license granted by the typeface owner(s).

The information in this file may be updated from time to time. The Unicode Consortium is not liable for errors or omissions in this excerpt file or the standard itself. Information on characters added to the Unicode Standard since the publication of Version 4.0 as well as on characters currently being considered for addition to the Unicode Standard can be found on the Unicode web site.

See http://www.unicode.org/pending/pending.html and http://www.unicode.org/alloc/Pipeline.html.

Based on ISCII 1988

Various signs

0981 ঁ BENGALI SIGN CANDRABINDU
0982 ং BENGALI SIGN ANUSVARA
0983 ঃ BENGALI SIGN VISARGA

Independent vowels

0985 অ BENGALI LETTER A
0986 আ BENGALI LETTER AA
0987 ই BENGALI LETTER I
0988 ঈ BENGALI LETTER II
0989 উ BENGALI LETTER U
098A ঊ BENGALI LETTER UU
098B ঋ BENGALI LETTER VOCALIC R
098C ঌ BENGALI LETTER VOCALIC L
098D ▨ <reserved>
098E ▨ <reserved>
098F এ BENGALI LETTER E
0990 ঐ BENGALI LETTER AI
0991 ▨ <reserved>
0992 ▨ <reserved>
0993 ও BENGALI LETTER O
0994 ঔ BENGALI LETTER AU

Consonants

0995 ক BENGALI LETTER KA
0996 খ BENGALI LETTER KHA
0997 গ BENGALI LETTER GA
0998 ঘ BENGALI LETTER GHA
0999 ঙ BENGALI LETTER NGA
099A চ BENGALI LETTER CA
099B ছ BENGALI LETTER CHA
099C জ BENGALI LETTER JA
099D ঝ BENGALI LETTER JHA
099E ঞ BENGALI LETTER NYA
099F ট BENGALI LETTER TTA
09A0 ঠ BENGALI LETTER TTHA
09A1 ড BENGALI LETTER DDA
09A2 ঢ BENGALI LETTER DDHA
09A3 ণ BENGALI LETTER NNA
09A4 ত BENGALI LETTER TA
09A5 থ BENGALI LETTER THA
09A6 দ BENGALI LETTER DA
09A7 ধ BENGALI LETTER DHA
09A8 ন BENGALI LETTER NA
09A9 ▨ <reserved>
09AA প BENGALI LETTER PA
09AB ফ BENGALI LETTER PHA
09AC ব BENGALI LETTER BA
= Bengali va, wa
09AD ভ BENGALI LETTER BHA
09AE ম BENGALI LETTER MA
09AF য BENGALI LETTER YA
09B0 র BENGALI LETTER RA
09B1 ▨ <reserved>
09B2 ল BENGALI LETTER LA
09B3 ▨ <reserved>
09B4 ▨ <reserved>
09B5 ▨ <reserved>
09B6 শ BENGALI LETTER SHA
09B7 ষ BENGALI LETTER SSA
09B8 স BENGALI LETTER SA
09B9 হ BENGALI LETTER HA

Various signs

09BC ় BENGALI SIGN NUKTA
• for extending the alphabet to new letters
09BD ঽ BENGALI SIGN AVAGRAHA

Dependent vowel signs

09BE া BENGALI VOWEL SIGN AA
09BF ি BENGALI VOWEL SIGN I
• stands to the left of the consonant
09C0 ী BENGALI VOWEL SIGN II
09C1 ু BENGALI VOWEL SIGN U
09C2 ূ BENGALI VOWEL SIGN UU
09C3 ৃ BENGALI VOWEL SIGN VOCALIC R
09C4 ৄ BENGALI VOWEL SIGN VOCALIC RR
09C5 ▨ <reserved>
09C6 ▨ <reserved>
09C7 ে BENGALI VOWEL SIGN E
• stands to the left of the consonant
09C8 ৈ BENGALI VOWEL SIGN AI
• stands to the left of the consonant

Two-part dependent vowel signs

These two-part dependent vowel signs have glyph pieces which stand on both sides of the consonant. These vowel signs follow the consonant in logical order, and should be handled as a unit for most processing.

09CB োা BENGALI VOWEL SIGN O
≡ 09C7 ে 09BE া
09CC ৌা BENGALI VOWEL SIGN AU
≡ 09C7 ে 09D7 ৗ

Various signs

09CD ্ BENGALI SIGN VIRAMA
= hasant (Bengali term for halant)
09CE ▨ <reserved>
09CF ▨ <reserved>
09D0 ▨ <reserved>
09D1 ▨ <reserved>
09D2 ▨ <reserved>
09D3 ▨ <reserved>
09D4 ▨ <reserved>
09D5 ▨ <reserved>
09D6 ▨ <reserved>
09D7 ৗ BENGALI AU LENGTH MARK

Additional consonants

09DC ড BENGALI LETTER RRA
≡ 09A1 ড 09BC ়
09DD ঢ BENGALI LETTER RHA
≡ 09A2 ঢ 09BC ়
09DE ▨ <reserved>
09DF য় BENGALI LETTER YYA
≡ 09AF য 09BC ়

Generic additions

09E0 ৠ BENGALI LETTER VOCALIC RR
09E1 ৡ BENGALI LETTER VOCALIC LL
09E2 ◌ৢ BENGALI VOWEL SIGN VOCALIC L
09E3 ◌ৣ BENGALI VOWEL SIGN VOCALIC LL

Digits

09E6 ০ BENGALI DIGIT ZERO
09E7 ১ BENGALI DIGIT ONE
09E8 ২ BENGALI DIGIT TWO
09E9 ৩ BENGALI DIGIT THREE
09EA ৪ BENGALI DIGIT FOUR
09EB ৫ BENGALI DIGIT FIVE
09EC ৬ BENGALI DIGIT SIX
09ED ৭ BENGALI DIGIT SEVEN
09EE ৮ BENGALI DIGIT EIGHT
09EF ৯ BENGALI DIGIT NINE

Bengali-specific additions

09F0 ৰ BENGALI LETTER RA WITH MIDDLE DIAGONAL
• Assamese
09F1 ৱ BENGALI LETTER RA WITH LOWER DIAGONAL
= BENGALI LETTER VA WITH LOWER DIAGONAL
• Assamese
09F2 ৲ BENGALI RUPEE MARK
09F3 ৳ BENGALI RUPEE SIGN
09F4 ৴ BENGALI CURRENCY NUMERATOR ONE
• not in current usage
09F5 ৵ BENGALI CURRENCY NUMERATOR TWO
• not in current usage
09F6 ৶ BENGALI CURRENCY NUMERATOR THREE
• not in current usage
09F7 ৷ BENGALI CURRENCY NUMERATOR FOUR
09F8 ৸ BENGALI CURRENCY NUMERATOR ONE LESS THAN THE DENOMINATOR
09F9 ৹ BENGALI CURRENCY DENOMINATOR SIXTEEN
09FA ৺ BENGALI ISSHAR

C: Nonstandard Word Example

□□□□□□ □□□□□ □□ □□/□/□□□□ □□ □□□□□□□□□□□□ □□□□□□□□ □□□□□□□ □□□□ □□ □□□ □□□ □□ STAR-Cineplex □ Movie □□□□□ □□□□□ □□□ □□□ □□□ □□□□□ □□□□□ Movie □□□□□ □□ □□□□□□ □□□ □□□□ □□□□ □□□□ □□□ □□□□□ □□□ Movie □□□□□ □□ □□□ □□□ □□□□□ Food-Court □ □□□□□□□□ □□□□ □□□□ □□□ □□□□□□□□ □.□□ □□□□□□ □□□□ □□□□□