Unified Praat Script Tools for Facilitating Korean ToBI Annotation[1)]

Kyuchul Yoon
(English Division, Kyungnam University)

1. Introduction

Corpus-based research has been one of the new driving forces not only in the area of natural language processing (NLP) but also in many linguistic fields. Many linguists look into text or speech corpora to find sentences or utterances relevant to their research. They use speech samples from the corpora to observe patterns, to build linguistic hypotheses and to test them. Engineers involved in the NLP business heavily depend on the corpora. They use the corpora to optimize and evaluate their systems.

Use of corpora is essential in some research areas because most corpora contain numerous textual or spoken sentences being used in real life. For example, the Penn Treebank (Marcus et al. 1993) text corpus contains 2,499

1) This work was supported by Kyungnam University Foundation Grant, 2006.

Key Words: Praat, script, Korean, ToBI, K-ToBI, annotation

stories from the Wall Street Journal which were syntactically annotated. Since these news articles were not intended for linguists in general, they contain real life expressions and useful collocations that can be exploited by computational linguists. The British National Corpus (British National Corpus 1991) contains a 100 million word collection of samples of written and spoken English. The Buckeye Speech Corpus (Pitt et al. 2005) contains speech recordings from 40 speakers in Ohio conversing freely with an interviewer.

One of the reasons that the text or speech corpora are useful is that they were annotated with information, which can be in the form of orthographic transcriptions, phonetic labeling, part-of-speech tags, and/or syntactic parsing. As more researchers examine linguistic phenomena with increasing attention to the prosodic hierarchy of a language, it is expected that there is an increased need for corpora which include prosodic annotation. For example, there has been growing interests in the behavior of segments in the prosodic hierarchy of Korean. In one study (Cho & Keating 2001), Korean stops were the main focus and they were examined in utterances annotated with the Korean ToBI prosodic transcription convention (Jun 2000). However, the study looked at the stop segments in utterances recorded only for that particular experiment. This is not to say that the experiment was flawed. Rather, the findings could be reconfirmed and supported with more follow-up studies performed with a large-scale K-ToBI annotated speech corpus.

Prosodically annotated corpora can also be used in speech synthesis. The importance of prosody, specifically of the fundamental frequency contour, segmental durations, phrasing information, etc. cannot be emphasized too much in the speech synthesis industry. In order to build an accurate phrasing model, for example, one can use a large-scale speech corpus

annotated with some kind of prosodic transcription convention. One study (Yoon 2006) built a 400-sentence speech corpus of Korean language sample annotated with K-ToBI and used it to build a model of phrasing that can be used in a text-to-speech synthesis system. If prosody has relevance in the linguistic patterning of segments and suprasegmentals, the importance of prosodically annotated speech corpora in modeling a language becomes self-evident.

Building an annotated corpus is a time-consuming process. It usually takes months, or even years of hard work. However, compared with the entire body of annotated corpora which has been compiled in the past, the number of corpora that are freely available is comparatively limited. Moreover, the tools used to build the corpora are rarely made public. Therefore, if one wishes to create a corpus similar to one which has already been built by someone else, many times one has to "reinvent the wheel". This is partly because such tools have usually been custom-built for a particular project or company. The Korean ToBI prosodic transcription convention is a very useful framework because, whichever software one may use, there is an established set of instructions that a labeler has to follow in order to annotate an utterance (Jun 2000). As long as the software allows one to develop tools that can be used in the corpus-building process, the annotated corpus will contain the same information as any other corpus using this system.

The challenge, then, is to develop such tools. Proprietary software such as PitchWorks(R) (Scicon R&D 2005) has limited capability in terms of allowing users to develop corpus-building tools. The data files are not plain texts, and can only be viewed within the software. The software only supports Windows and Mac operating systems and is therefore closed to the Linux/Unix community. All of these "closedness", tool development and

platform compatibility problems disappear with open-source programs such as Praat (Boersma 2005) and WaveSurfer (Center for Speech Technology 2005). For example, Praat has a built-in scripting language that allows users to write scripts to meet their own individual needs. These script tools can be used to automate repetitive tasks in building corpora. All of the data files produced by the program are plain text files that can be viewed and edited with any text editor. The program supports Windows and Mac as well as Linux/Unix operating systems. Data files created in a Linux or Unix environment can be then opened in a Windows working environment, and vice versa. The source codes used to create Praat are open and can be edited if one otherwise has the capability and the need to do so. Open-source projects are becoming of interest to more and more people because, unlike propriety software which may disappear along with the company that has created it, the project does not depend on merely a few select people, but rather, relies upon an entire widespread community of people who are willing to devote their time and energy to such projects. In brief, open-source programs can similarly be said to offer all of the benefits of an open-source project.

For these reasons, more people are using open-source programs to design and build corpora today than ever before. Communities of people who use the open-source programs typically exchange ideas, scripts and information in order to improve them. The Praat scripts which have been introduced in this paper are the ones that were originally designed to build the OSU Korean Talkbank speech corpus used in Yoon (2006). The original Praat scripts were modified and extended to produce a unified set of scripts that one can use to build a large-scale K-ToBI labeled speech corpus. The Praat scripts are 'unified' in the sense that, once the text part of the corpus is romanized with the help of a non-Praat script, the entire corpus-building

process can be completed with the scripts introduced below. This process includes steps ranging from tokenizing texts and speech into sentence-sized chunks to evaluating inter-labeler agreement. The scripts help labelers speed their annotation either by automating some steps, e.g., tokenizing a paragraph into punctuation-delimited sentences, or by eliminating repetitive tasks, e.g. opening and closing files, creating default label files, or assigning default tone labels to be corrected by the labelers. The author intends to distribute these scripts under the open-source terms of GNU General Public License (GPL) (Free Software Foundation Inc. 1991). Users of these scripts can freely modify them to meet their needs in conducting research.

2. Praat scripts for Korean ToBI annotation

For the operational purpose of this work, the process of building a speech corpus annotated with the K-ToBI convention was classified into five steps. In addition to these five essential steps, some maintenance work may also be further performed on a corpus, such as normalizing the tokenized sound files or correcting errors made by labelers.

- Tokenize a text paragraph into component sentences delimited by punctuation marks.
- Tokenize a paragraph-long sound recording into matching utterances.[2)]
- Create an annotation file (.TextGrid), align by word and assign default labels.

2) The unit of tokenization can be larger than a sentence, e.g. a paragraph, in which case a slight modification of the scripts for the next step would be necessary so that appropriate default labels can be assigned.

- Labelers perform the actual K-ToBI annotation.
- Evaluate inter-labeler agreement.

The current version of Praat cannot handle two-byte characters such as Korean hangul. Thus, it was assumed that the textual part of the corpus had been already romanized by some non-Praat script[3] (see footnote 2). It was also assumed that the romanized hangul would preserve the orthographic syllabification information inherent in the hangul writing system. One way to do it was to insert a hyphen between the end of one orthographic syllable and the beginning of the following orthographic syllable. For example, the name for the Korean writing system 한글, which has two orthographic syllables, was romanized as *han-gul*, not as *hangul*. The reason we chose to preserve the orthographic syllabification information in the romanized hangul is that it would make things easier when default tone labels were assigned during the third step. Depending on the number of syllables present, the default phonetic tone labels for an Accentual Phrase (AP) can vary from two to four units. For example, if the number of syllables comprising an AP is four or more, it is highly likely that the phrase will have the default tone labels of, e.g. L +H L+ Ha. The actual realization of the surface tone pattern can be changed by the labeler, which is why the preliminary tone labels are labeled 'default'.

Inserting these 'default' tone labels automatically by the Praat script is one way in which these script tools save time in building the speech corpus. From the experience of building the OSU Talkbank, it was realized

3) The romanization code that we used was "han2phon.c" and it was originally written by Nick Cipollone, an OSU Linguistics graduate currently working at Microsoft Corporation, and later modified by Prof. Chris Brew at the OSU Linguistics Department.

that it took a considerable amount of time to manually open a sound file, create a TextGrid file and insert somewhat repetitive surface tone labels in the phonetic tone tier. It was also noticed that it took much less time to delete a tone label than to insert it, or to change the position of a tone label, than to insert it. This gave us the idea of inserting 'default' tone or boundary index labels into the TextGrid file. Then the labelers can simply move the default labels around, or remove them if necessary. In a carefully read speech style, an orthographically space-delimited word group, or *eojeol*, tends to be realized as one AP. It therefore makes sense to insert a 'default' phonological tone label at the end of each eojeol. When two or more eojeol combines to form a bigger AP, the labeler can merely delete the intervening default tone label. A sentence usually ends with an Intonational Phrase (IP), whose common boundary tone label is HL%. As a result of this tendency, this tone can be assigned as a default tone at the end of every sentence. The break indices associated with the usual AP and IP are numbers 2 and 3. These values can also be assigned as the default values to the break index tier of the TextGrid file. The strategy of using 'default' labels was justified with English ToBI labeling (Syrdal et al. 2001), which found that automatic assigning of 'default' labels speeded manual labeling, and it did not exhibit any significant bias effect on label assignment.

Moreover, time can be saved in other steps as well. When building a large-scale speech corpus, it is usually the case that the labelers provide some kind of an orthographic transcription to each sentence. If there is already a text version of the corpus, it is usually chunked or tokenized into sentences. The spoken version is also tokenized into matching utterances. Of course, this cannot always be the case. However, tokenizing the original long sound file is recommended because it is much easier to handle small

sound files than large bulky files. When choosing to perform the tokenizing, a Praat script can perform all of the repetitive tasks involved in the step. Labelers need only to identify the start or end of each sentence. The script will cut the identified segment and save it as a sentence-sized sound file on the labeler's computer hard drive. Furthermore, the script can be instructed to insert a labeling tier, e.g. a "sentence" tier, in the TextGrid file, so that the original sentence stays in one of the labeling tiers. As suggested above, it is much easier to delete information than to retrieve deleted information. Therefore, there is no harm done in adding more information to the TextGrid file. After all, the K-ToBI convention encourages labelers to add information if necessary.

One essential step necessary in building a large-scale corpus is to assess the degree of agreement or consistency among the annotators involved in the project. If there is not consistent agreement above a certain level, the corpus cannot be reliably used by other researchers. One way to rate inter-labeler agreement is to compare the *transcriber-pair-word* (Pitrelli et al. 1994). This is a comparison of the labels that two particular transcribers assigned to one particular word or word boundary. It thus analyzes the combinatorics of picking two transcribers out from among all of the transcribers who participated in the project. If there were *n* number of transcribers, then the combinatorics of picking two transcribers, ignoring their order, would be nC2 = n*(n-1)/2. In other words, two transcribers are compared at a time until all such possible transcriber pairs have been exhausted. The resulting percent agreement rate will be based on this number. If there were 10 transcribers, the 100% agreement would be reached for a particular label if all 10*(10-1)/2 =45 transcriber-pairs assigned the same label, i.e. 45/45*100 = 100%. If one transcriber did not agree with the rest, then the transcriber-pairs would be 9*(9-1)/2 = 36, and the

agreement would be 36/45*100 = 80%. As Pitrelli et al. (1994) claims, this is a more strict way of calculating inter-labeler agreement, because without the notion of transcriber-pair-words, the 80% agreement rate would have corresponded to a 90% agreement, because nine transcribers out of ten agreed. When analyzing merely a couple of sentences picked out for calculating inter-labeler agreement, manual calculation could also work. If, however, dozens of sentences were involved, doing this manually would be prone to introduce human errors. Therefore it is more desirable to use a script for evaluating inter-labeler agreement.

The following subsections elaborate on what the Praat scripts do in each of the five steps given above. The last subsection introduces two useful scripts that can be used in the corpus-building process. The author assumes that the reader has basic knowledge about Praat. The text versions of the scripts are given in Appendix and their file versions are available on the author's website.

2.1. Tokenizing a text paragraph into sentences

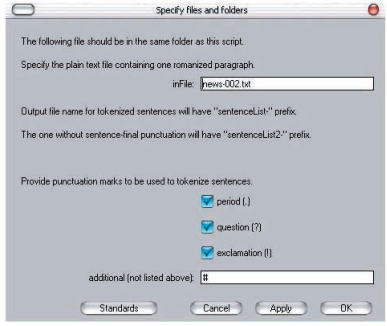
The first thing to do after the romanization of the hangul text is to tokenize a paragraph into component sentences delimited by sentence-final punctuation marks. Note that the paragraph should not contain any line breaks. In other words, the paragraph is a single long line containing many component sentences. If there is more than one paragraph in the input text file, the script will display an error message, in which case the user has to make sure that the input file has only one paragraph. The tokenizing script takes the paragraph as its input and puts out a column of component sentences. Each row of the column is separated by a line break. The input and its output of the script will look as follows after the execution.

<Input: Before execution>

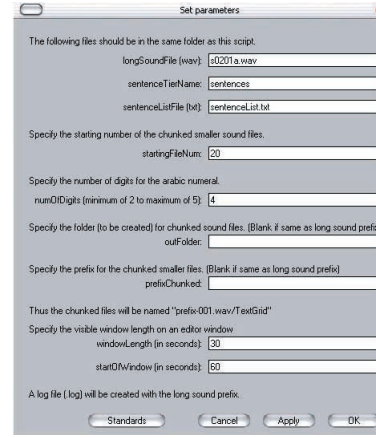
Sentence 1. Sentence 2? Sentence 3! Sentence 4. Sentence 5?

<Output 1: After execution>	<Output 2: After execution>
Sentence 1.	Sentence 1
Sentence 2?	Sentence 2
Sentence 3!	Sentence 3
Sentence 4.	Sentence 4
Sentence 5?	Sentence 5

<Figure 1> shows the dialog box that pops up after the execution of the tokenizing script. It asks the user to type in the name of the input text file. Note that the file should be a plain text file that can be edited with any text editor such as “Notepad” in the Windows environment. As the dialog box shows, the output file containing a column of component sentences will



<Figure 1> Dialog box of the text tokenizing script.



<Figure 2> Dialog box of the sound tokenizing script.

have the same name as the input file except that it will be prefixed with "sentenceList-". For example, for the input file "news-002.txt", the output file name should be "sentenceList-news-002.txt". The script will then produce another output file without the sentence-final punctuation, this time with the prefix "sentenceList2-". Of course, this prefix can be changed in the script by replacing it with whatever the user may desire. As the check boxes indicate in the dialog box, the script uses at least three types of sentence-final punctuation; a period, question mark or exclamation point. Users can also provide an additional symbol in the blank, if desired. If there is a symbol that the script has to use in the tokenization, this is the place to specify it.

The script proceeds by reading the entire paragraph as one long line, identifying the location of the punctuation marker that appears first, and then cutting and storing in a string array variable whatever text occurs previous to the first punctuation mark, including the punctuation mark itself. It will then output the first component sentence into the output text file. The script keeps repeating this process with whatever text remains from the previous step. The commands that play a major role in the process are the functions, *index( )*, *left( )*, and *right( )*, and the *while ~ endwhile* loop. The *index( )* function deals with two string arguments. The first one in this case would be the long string of component sentences and the second one would be one of the punctuation marks. The function obviously identifies the first occurrence of the punctuation mark. Once it is known where the first sentence ends, the first component sentence can then be extracted from the long line of sentences by using the *left( )* function, which also deals with two arguments; one is a string and the other is the number of characters to be extracted from the left end of the string argument. What remains from this step is stored in another string variable

by using the *right( )* function, which will serve as the input for the next round of the *while* loop.

2.2. Tokenizing a paragraph-long sound recording into matching utterances

What the user needs to do in tokenizing the paragraph-long sound file into sentence-sized utterances is to identify the start and end of each sentence by click-dragging over each sentence in the editor window, and pressing the "Continue" button of the small pop-up dialog box created by the script command *pause*. The script will also display an information window containing a column of all of the component sentences. The user only needs to check each sentence in the information window, select the corresponding utterance in the editor window, and push the "Continue" button in another pop-up dialog box in order to continue to the next sentence.

As <Figure 2> shows, the script takes as its input the two files, one for the paragraph-long sound file and the other for the text file with tokenized sentence. Note that the script is about to create an annotation file whose first labeling tier will be named "sentences". If the user does not like this tier name, it can be changed to something like "tokenizedSentence", etc. The tokenized utterance files will be named after the paragraph-long sound file, except that they will be sequentially numbered. For example, if the long sound "s0201a.wav" contained dozens of utterances, the tokenized ones will be named "s0201a-01.wav", "s0201a-02.wav", "s0201a-03.wav", etc. If the long sound file contained hundreds of utterances, the tokenized ones should be named "s0201a-001.wav", "s0201a-002.wav", "s0201a-003.wav", etc. The number of digits, e.g. 2 for the former and 3 for the latter, can be

specified in the *numOfDigits* field. It can take 2 to 5 digits. The script can thereby tokenize workloads ranging from merely dozens of utterances, i.e. 01 ~ 99, to hundreds of thousands of utterances, i.e. 00001 ~ 99999.

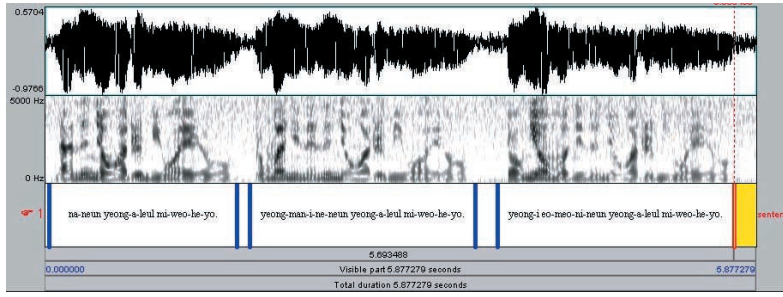
If the user were to stop after tokenizing 987 utterances and take the day off, he or she could then resume the tokenizing process upon resuming work at any time starting with the 988th utterance. This process of break and resumption can be specified in the *startingFileNum* field. The position of the cursor in the editor window can be restored by entering the location in seconds in the *startOfWindow* field. The user does not have to memorize the cursor position. It can be retrieved from the log file that was created in the same directory as the script. The user can also specify the size of the window to display with the *windowLength* field. With a bigger monitor, the user can similarly specify a wider window length. If the user leaves the *outFolder* and *prefixChunked* field blank, the output folder and the tokenized utterance files will have the same name as the long sound file, which is recommended in order to avoid any possible file name confusion.

With the tokenization in progress, the user will notice that the script keeps adding sentences to the sentence tier of the TextGrid label file that is displayed simultaneously with the long sound file (See <Figure 3>). If either the last sentence is reached or the user has not selected anything, i.e. signaling that the user has finished with the tokenizing or that the user wants to take the day off, the script will display a small pop-up question dialog box, and ask the user if she or he wants to do the actual cutting of the labeled utterances, and save them into smaller sound files. At the same time, the TextGrid files with only one tier containing each sentence will be saved in the same output folder. The text or sound tokenizing process can be repeated as many time as there are text files, each containing a paragraph. If there were 30 text files, i.e. 30 paragraphs, then there would

be 30 subfolders after the tokenization. Each subfolder would contain tokenized sound/TextGrid files which would be component sentences of each paragraph.

2.3. Create an annotation file, align by word and assign default labels

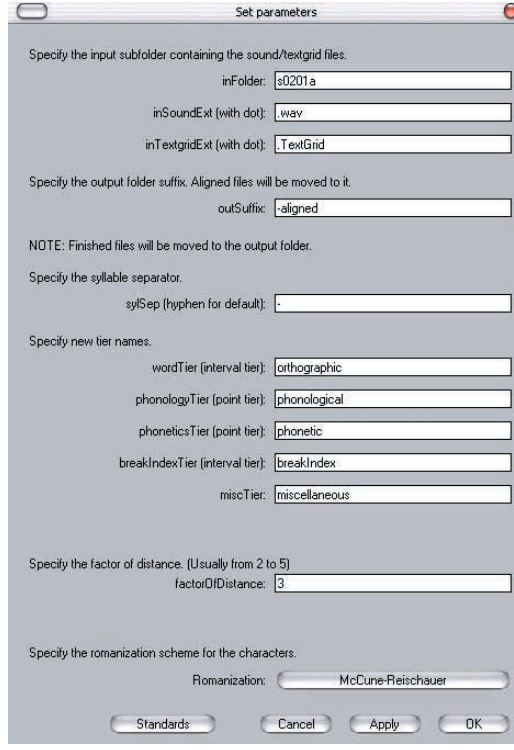
Strictly speaking, the annotation file, i.e. the TextGrid file, was already created in the immediate preceding step where the sound tokenizing script chopped the paragraph-long sound files into sentence-sized sound files, along with their matching TextGrid files. However, the current step can be said to create the annotation file because this is where the K-ToBI labeling tiers such as the phonological tone tier, phonetic tone tier, break index tier, etc. are added to the existing TextGrid file. As the first blank field in <Figure 4> show, the script asks for the name of the input folder where the tokenized sounds and TextGrids are. In the next two fields, it asks for the file extensions of the files because it will only work with those types of files in the input folder. Then it asks for the suffix that will be used to create the output folder where the aligned TextGrid files are to be stored. For



<Figure 3> The sound tokenizing script keeps adding sentences to the “sentences” tier.

example, if the input folder with the TextGrids is designated “s0201a”, then the output folder will be named “s0201a-aligned” if the user accepts the default suffix provided in the fourth blank field.

Note that the script does not modify the existing TextGrids and save them in the same folder, but rather creates an additional folder that will house the modified files. This is one of the design principles of the unified script set. We observed this principle because we had learned from our own previous experiences that it was generally preferable to “insert” than to “overwrite”

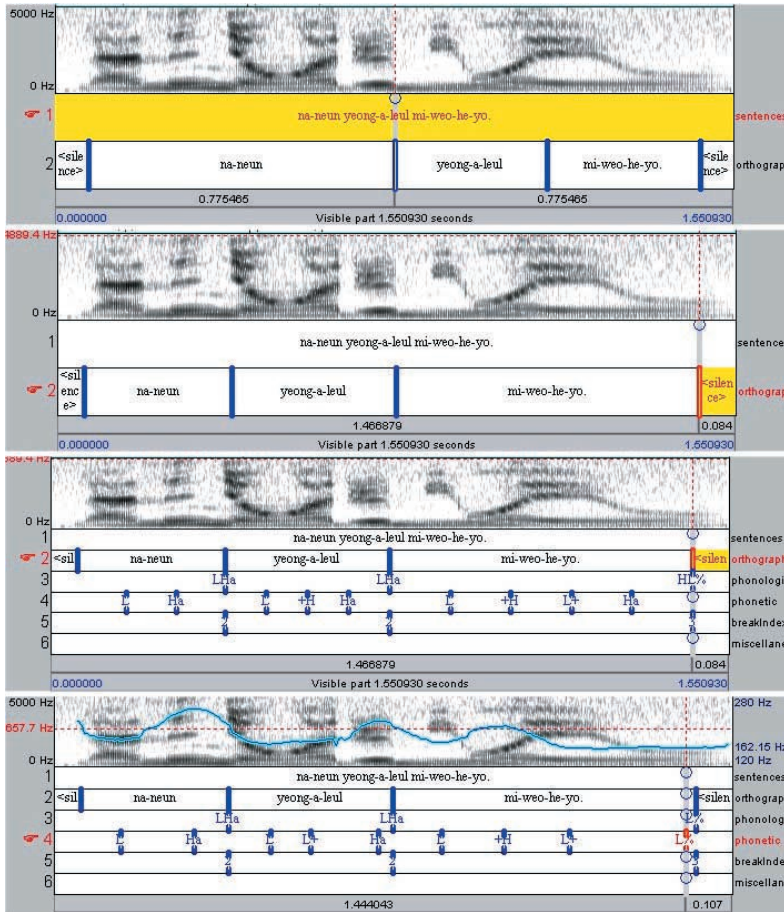


<Figure 4> Dialog box of the script for creating and aligning an annotation file, and assigning default labels.

information. Once overwritten, old information, which could be of critical help later on, could not be retrieved. Since the storage space has become increasingly cheaper over the years, it is always a good idea to keep the original data in a folder, while putting the newer version into a new folder.

The default symbol used to separate orthographic syllables in the romanized data is the hyphen. This is given in the *sylSep* field. The next five fields specify the names of the labeling tiers to be added to the existing TextGrid files. Once the script is run, it automatically tokenizes the sentence in the first labeling tier into its component space-delimited words, create an additional "word" tier, and puts them into their own separate compartments called intervals in the word tier. The *factorOfDistance* fields specifies how much the tokenized words are crowded toward the right hand side of the orthographic word tier before being manually aligned by a labeler. The factor takes an integer value from 2 to 5: "2"designating the most density bias toward the right, and "5", the least right-handed density bias of the tokenized words. The labeler must then move the boundaries of the word intervals to their appropriate place according to the waveform and spectrogram of the sound file. If the boundary to be moved were placed by the script well before, i.e. to the left hand side of, the matching acoustic landmark on the spectrogram, the labeler would first have to move all of the boundaries to the right of the current boundary to make room for the current boundary to be able to move to its appropriate position. This is very inefficient and is a waste of time. Therefore a way is needed to "push" the tokenized words toward the right hand side of the tier so that labelers can retrieve one word at a time. Using the *factorOfDistance* is one way to accomplish this "pushing" in the editor window.

The alignment of the tokenized words is the only thing that a labeler would have to do manually. The script would perform all of the rest of the



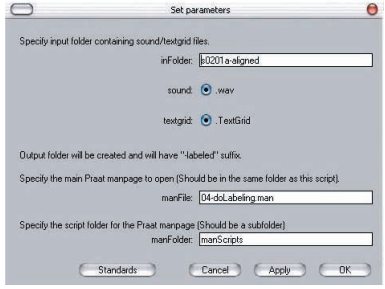
<Figure 5> What the 'default' label assigning script does. It first tokenizes the sentence into its component words (top panel). The labeler manually aligns the words (second from the top). The script assigns 'default' labels after creating four additional labeling tiers (second from the bottom). The labeler manually corrects the 'default' labels, completing the K-ToBI labeling for the sentence (bottom panel). Model labeling from (Jun 2000).

remaining processes automatically, from adding four additional labeling tiers to placing default boundaries in the tiers according to the K-ToBI labeling convention (See <Figure 5>). Specifically, following the manual alignment of the component words, the script will add AP boundaries with the default *LHa* label at the end of each word, default phonetic tone tier labels according to the identity of the first syllable onset and the number of component syllables of each word, and the default break indices; 2 for an AP and 3 for the end of the sentence. The identity of the first syllable onset can be changed depending on the romanization scheme used in the hangul romanization. This is done by selecting the right scheme in the *Romanization* drop-down menu at the bottom of the dialog box. When all of the default labels have been assigned, the script will pause, asking the user to check the newly-created TextGrid, and to press "Continue" in order to save it to the new output folder specified by the user. The same process may then be repeated for the next file in the input folder.

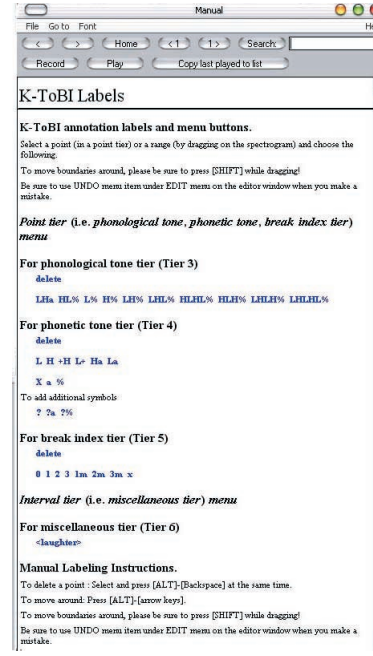
2.4. Perform the annotation

In this step, the labeler does the most important job in building the corpus, i.e. listening to the utterance and correcting the default labels which have been assigned in the previous step. This is where keen judgment about the actual tone/break index labels is required by the labelers. Since most of the labels that a labeler would require in this step have already been inserted by the previous script, the labeler needs only to delete unnecessary labels, and, in some cases, to correct the wrong ones.

In this step, there are two types of scripts involved; one is a Praat script (see <Figure 6>) and the other is a Praat manpage (see <Figure 7>). A Praat manpage is a hyperlink-style document format used in the Praat manuals.



<Figure 6> Dialog box of the labeling script.



<Figure 7> Praat manual page containing script links (in blue).

Hyperlinks (in blue) can take one to other manpages, or perform linked Praat scripts. The hyperlinks used in this step are all Praat scripts that perform actions to the TextGrid file in the editor window. These scripts reside in a subfolder specified in the dialog box in <Figure 6>.

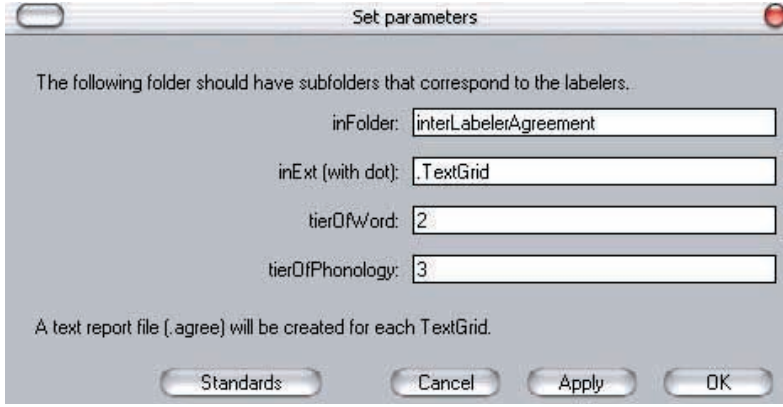
Once the script is run, it will also bring up a Praat manpage given in <Figure 7> along with an editor window with the first sound/TextGrid pair in the input folder. If the labeler wished to delete a particular boundary in the phonological tone tier, he or she would have to select the boundary by clicking on it, and then select the *delete* link provided under the "For phonological tone tier" section. The labeler can replace an existing label

with one of the labels provided in each of the section in the manpage. This is done by selecting an existing label, and clicking on one of the labels in the manpage. Of course, the labeler can manually delete, or correct any existing labels in the editor window. Manual labeling instructions are given at the bottom of the manpage.

In the phonetic tone tier, in addition to the usual editing procedures explained above, the labeler can delete, insert, or move the surface tone labels according to his, or her judgment. The labeler can do the same editing in the break index tier. Finally, the miscellaneous tier, which is an *interval tier*, requires special attention. Since it is not a point tier, it requires two boundaries in order to create one compartment or interval. The labeler can specify the beginning, and end of the interval by selecting an area over the editor window. This is done by clicking on the beginning of the area, and dragging the mouse to the end of the area. Click on the <laughter> link on the manpage, and the interval in the miscellaneous tier will be created for the labeler by the embedded script. When the actual labeling is done for the TextGrid, the labeler can press “Continue” on the pop-up dialog box, and the script will store the newly-created TextGrid file into a different output folder whose name has the “-labeled” suffix after the input folder name.

2.5. Evaluate inter-labeler agreement

As explained above, evaluating inter-labeler agreement is a very important step in building any large-scale corpus, and we followed the strict method proposed in Pitrelli et al. (1994). As the dialog box in <Figure 8> shows, there should be a folder which has subfolders. Each subfolder should contain TextGrids labeled by each labeler. Of course, the set of



<Figure 8> Dialog box of the script for evaluating inter-labeler agreement.

TextGrids should be the same for all labelers. Once the script is run, it will look for each file in the subfolders, compare each label points, count transcriber-pair-words, and calculate the overall percent value of the degree of agreement. The user needs only to designate the orthographic word tier, and the phonological tone tier. As of the time of this publication, with regard to the current version of the script, the degree of agreement in the phonetic tone tier has not as yet been calculated.

For each TextGrid, the script will create a plain text file containing all of the labels each of the labelers assigned for a particular word boundary, along with their transcriber-pair-word counts, and agreement for that TextGrid. If there were 12 subfolders or labelers used in the inter-labeler agreement evaluation, there would be 12 plain text files with .agree file extension in the current folder. Each text file will contain the same kind of information except that the accumulated percent agreement will be given in the last file. A sample text file is given below (TPW stands for transcriber-pair-word).

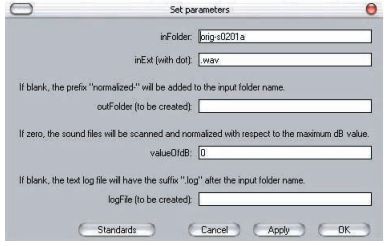
WORD	Yeong-i-ga	o-myeon	na-neun	gan-da	
Labeler 1	NONE	LHa	LHa	HL%	
Labeler 2	NONE	LHa	LHa	HL%	
Labeler 3	NONE	LHa	LHa	HL%	
Labeler 4	NONE	LHa	LHa	HL%	
Labeler 5	NONE	LHa	LHa	HL%	
Labeler 6	NONE	LHa	LHa	HL%	
Labeler 7	NONE	LHa	LHa	HL%	
Labeler 8	NONE	LHa	LHa	HL%	
Labeler 9	NONE	LHa	LHa	HL%	
Labeler 10	NONE	HL%	LHa	HL%	
Labeler 11	NONE	HL%	LHa	HL%	
Labeler 12	NONE	HL%	LHa	HL%	
ActualTPW	66	**39**	**66**	**66**	**234**
ReferenceTPW	66	66	66	66	264
234 out of 264	88.6%				

(39 TPW for “o-myeon” because 9 Lha’s and 3 HL%’s, i.e. 9*(9-1)/2 + 3*(3-1)/2 = 39)

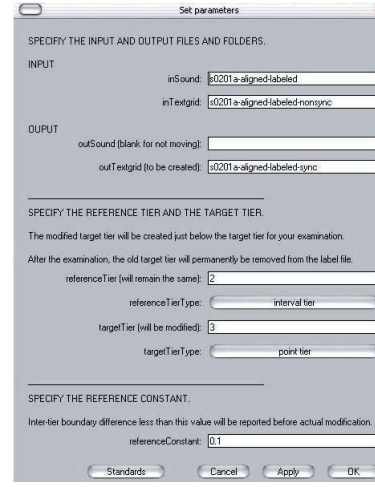
2.6. Miscellaneous scripts

When it is necessary to normalize paragraph-long sound files, the normalizing script (see <Figure 9>) can be used. It will scan input sound files for maximum dB value and normalize them with respect to the value. If the user has specified a particular dB value, it will use that value instead. The script will also produce a text log file that contains information with regard to the intensity values before, and after the normalization.

When a labeler accidentally moves a boundary that is not supposed to be moved, or inserts a boundary into a place slightly out of sync with



<Figure 9> Dialog box of the normalizing script.



<Figure 10> Dialog box of the boundary-synchronizing script.

boundaries in the other tiers, he or she can undo that action in the editor window. However, if this mistake were not corrected all the way through the final labeling step, it can be corrected by using the boundary-synchronizing script. <Figure 10> shows the parameters the user can set. Users can specify the reference tier that the script will use as the reference, and the target tier that the script will synchronize. The *referenceConstant* field at the bottom of the dialog box indicates the threshold of the discrepancy between the boundaries across the two neighboring tiers. If the constant were 0.1, the script would synchronize the boundaries if the inter-boundary distance across the two tiers is less than 0.1 seconds. The constant can therefore be determined on an empirical basis. Once the constant is determined, the script will output the synchronized TextGrids in the user-specified subfolder.

3. Discussion

This paper introduced a set of Praat scripts to help speed K-ToBI annotation. The process of building a large-scale speech corpus was classified into five steps: tokenization of the text part of the corpus into component sentences, tokenization of a paragraph-long sound file into matching utterances, creation of TextGrid files followed by manual alignment of the tokenized words and assignment of default tone and break index labels, actual annotation by the labelers, and evaluation of inter-labeler agreement. Praat scripts were designed to save time in each of these steps either by making some of the processes automatic, or by eliminating repetitive tasks. The script set can be said to be 'unified' in the sense that once the hangul text is romanized, the entire corpus-building process can be handled with the use of the scripts.

The scripts will be distributed under the terms of the GNU General Public License (GPL) (Free Software Foundation Inc. 1991), in the hope that researchers use and modify them to suit their needs, and will in turn publish them in order to encourage and facilitate further research. It is also hoped that more annotated corpora will be freely available to the academic community. Increasing numbers of researchers are beginning to realize that corpus study is important not only to computational linguists, but also to theoretical linguists as well. Linguists who may not be particularly interested in practical applications of the corpora can still look for linguistic patterns which may be yet of interests to them in the corpora, and use them for theoretical study. The quality of the materials which such theoretical researchers may find in the corpora could be particularly useful in that such materials were not recorded or collected for the sole purpose of the

researchers' specific areas of theoretical study. The evidence linguists find in the corpora could thereby make their claims stronger than those they find in the materials obtained in recording facilities. The Buckeye Speech Corpus (Pitt et al. 2005) is one such example. It contains high-quality recordings from 40 speakers from Columbus OH conversing freely with an interviewer. The corpus was orthographically transcribed and phonetically labeled. Researchers can study the speakers in the corpus in terms of phonetic or phonological phenomena, regional variations, etc.

Quality corpora can contribute to the advance of applied and theoretical speech sciences. However, a lot of corpora created by human annotators are bound to contain human errors. The speech corpora that will be created by our scripts can contain errors. Just as slips of the tongue made by language-acquiring children shed light on the developmental aspects of the child language acquisition, errors in the corpora could be a valuable asset in the speech science. Studies (Dickinson & Meurers 2005a, 2005b) show that error-detecting techniques can be developed for speech corpora. Therefore, it appears important that the academic community publishes their work, even if it contains errors, for others to study and use it. One advantage of using an open-source program such as Praat is that it is much easier to develop scripts for correcting errors. This is especially so if the labeler has made a particular kind of mistake across his or her entire annotation. Once the nature of the mistake is revealed, it only requires a minimum of time and effort to write a script, and correct the error. The boundary-synchronizing script introduced above is one such example.

The file versions of all the Praat scripts and demo files can be obtained from the author's website. Future versions of the scripts will also be posted on the website. Bug reports and advice on improving the scripts are always welcomed. Current versions are not adequate in terms of generating

sufficient error messages to the user, which we hope will be improved in future versions.

References

Boersma, P. (2005), “Praat, a system for doing phonetics by computer”, Glot International, Vol.5, 9/10, 341-345.

British National Corpus (1991), “Written corpus design specification”, BNC Working Paper 08.

Centre For Speech Technology (CTT) (2005), WaveSurfer (Software), KTH in Stockholm, Sweden.

Cho, T. & Keating, P.A. (2001), “Articulatory and acoustic studies on domain-initial strengthening in Korean”, Journal of Phonetics, Vol.25, 155-190.

Dickinson, M. & Meurers, W.D. (2005a), “Detecting annotation errors in spoken language corpora”, Proceedings of the Special session on treebanks for spoken language and discourse at the 15th Nordic Conference of Computational Linguistics (NODALIDA-05). Joensuu, Finland.

Dickinson, M. & Meurers, W.D. (2005b), “Detecting errors in discontinuous structural annotation”, Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05), 322-329, Ann Arbor, Michigan, USA.

Free Software Foundation Inc. (1991), GNU General Public License, Version 2, June.

Jun, S. (2000), “K-ToBI (Korean ToBI) labeling conventions: Version 3”, Speech Sciences, Vol. 7, 143-169.

Marcus, MP., Marcinkiewicz, M.A. & Santorini, B. (1993), "Building a large annotated corpus of English: the Penn Treebank", Computational Linguistics, Vol.19, Issue 2, 313-330.

Pitrelli, J., Beckman, M. & Hirschberg, J. (1994), "Evaluation of prosodic transcription labeling reliability in the ToBI framework", Proc. ICSLP, 123-126, Yokohama, Japan.

Pitt, M., Johnson, K., & Hume, E. (2005), "The Buckeye Corpus of Conversational Speech: Labeling conventions and a test of transcriber reliability", Speech Communication, Vol.45, 90-95.

Scicon R&D (2005), PitchWorks (Software), California, USA.

Syrdal, A.K., Hirschberg, J., McGory, J., & Beckman, M. (2001), "Automatic ToBI prediction and alignment to speed manual labeling of prosody", Speech Communication, Vol.33, 135-151.

Yoon, K. (2006), "A prosodic phrasing model for a Korean text-to-speech synthesis system", Computer Speech and Language, Vol.20, Issue 1, 69-79.

Appendix (Praat scripts)

* NOTE: All the scripts below are distributed under the terms of GNU Public License. Online file versions are available on the author's web site. For instructions, consult section 2. Praat scripts for Korean ToBI annotation.

1. Tokenize a paragraph by sentence

```
############################################################################
# tokenizeParagraphBySentence.praat (Written by Kyuchul Yoon  kyoon@kyungnam.ac.kr)
############################################################################
# GNU General Public License
# Copyright (C) Kyuchul Yoon, English Division, Kyungnam University, South Korea.
```

```
##############################################################################
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
# This program is distributed in the hope that it will be useful,but WITHOUT
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
# FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more details.
##############################################################################
# This license message was omitted in the other scripts below to save space.
##############################################################################
# Before you run, check that you have the following ready.
# (1) A plain text file containing one (only one!) romanized paragraph with intact
#     sentence-final punctuation marks.
#-----------------------------------------------------------------------------
# This script will tokenize the paragraph into punctuation-separated sentences.
# User can specify which punctuation marks to use.
# After the tokenization, users can optionally remove sentence-final punctuation.
# Thus, two output files can be produced, one with the sentence-final punctuation,
# and the other without the punctuation.
##############################################################################
form Specify files and folders
    comment The following file should be in the same folder as this script.
    comment Specify the plain text file containing one romanized paragraph.
    word inFile news-002.txt
    comment Output file name for tokenized sentences will have “sentenceList-” prefix.
    comment The one without sentence-final punctuation will have “sentenceList2-” prefix.
    comment
    comment Provide punctuation marks to be used to tokenize sentences.
    boolean period_(.) 1
    boolean question_(?) 1
    boolean exclamation_(!) 1
    word additional_(not_listed_above) #
endform

# Create the output file name.
outFile$ = “sentenceList-” + inFile$
```

```
# Get the punctuation marks to use.
if period = 1
    puncMark$ = "."
    # Read the paragraph text file.
    Read Strings from raw text file... 'inFile$'
    Rename... inFileObj
    numParagraphs = Get number of strings
    # If there's more than one paragraph, give a warning message.
    if numParagraphs > 1
        exit Terminating... More than one paragraph!
    endif
    # Load the paragraph
    paragraphText$ = Get string... 1
    # And tokenize the paragraph by the sentence period.
    indexOfPeriod = index(paragraphText$,puncMark$)
    lengthOfParagraph = length(paragraphText$)
    iSentence = 0
    while indexOfPeriod <> 0
        iSentence = iSentence + 1
        sentence'iSentence'$ = left$(paragraphText$,indexOfPeriod)
        paragraphText$ = right$(paragraphText$,(lengthOfParagraph-indexOfPeriod))
        lengthOfParagraph = length(paragraphText$)
        # Remove any leading spaces iteratively.
        existSpace = startsWith(paragraphText$, " ")
        while existSpace = 1
            paragraphText$ = right$(paragraphText$,(lengthOfParagraph-1))
            lengthOfParagraph = length(paragraphText$)
            existSpace = startsWith(paragraphText$, " ")
        endwhile
        # Then continue searching for the next sentence.
        indexOfPeriod = index(paragraphText$,puncMark$)
        lengthOfParagraph = length(paragraphText$)
    endwhile
    # Store the last paragraph, if any, to the array variable.
    iSentence = iSentence + 1
    sentence'iSentence'$ = paragraphText$
    # Output the tokenized sentence array variable to a temp file for debugging purpose.
    for i to iSentence
```

```
            dummy$ = sentence 'i'$
            fileappend 'outFile$' 'dummy$' 'newline$'
        endfor
endif
select Strings inFileObj
Remove

if question = 1
        puncMark$ = "?"
        call tokenizeProc 'puncMark$'
endif

if exclamation = 1
        puncMark$ = "!"
        call tokenizeProc 'puncMark$'
endif

if length(additional$) = 1
        puncMark$ = additional$
        call tokenizeProc 'puncMark$'
endif

# Optional procedure for removing sentence-final punctuation.
pause Remove sentence-final punctuation too? Continue for yes, Stop for no.
# Create a filename.
outFile2$ = "sentenceList2-" + inFile$

Read Strings from raw text file... 'outFile$'
Rename... outFileObj
numLines = Get number of strings
for j to numLines
        select Strings outFileObj
        lineText$ = Get string... j
        lengthOfLineText = length(lineText$)
        if length(additional$) <> 0
            if (endsWith(lineText$, ".") or endsWith(lineText$, "?") or endsWith(lineText$, "!")
            ... or endsWith(lineText$,additional$))
                lineText$ = left$(lineText$,(lengthOfLineText-1))
```

```
            fileappend 'outFile2$' 'lineText$' 'newline$'
        endif
    else
        if (endsWith(lineText$, ".") or endsWith(lineText$, "?") or endsWith(lineText$, "!"))
            lineText$ = left$(lineText$,(lengthOfLineText-1))
            fileappend 'outFile2$' 'lineText$' 'newline$'
        endif
    endif
endfor
select Strings outFileObj
Remove

procedure tokenizeProc punc$
    # Read the temporary output text file
    Read Strings from raw text file... 'outFile$'
    Rename... outFileObj
    numLines = Get number of strings
    # Delete the text file for it will be recreated below.
    filedelete 'outFile$'
    # For each pseudo-sentence, check if it has any of the demarcating symbol.
    for i to numLines
        select Strings outFileObj
        lineText$ = Get string... i
        indexOfPunc = index(lineText$,punc$)
        lengthOfLineText = length(lineText$)
        # If it has no target symbol, just print it out.
        if indexOfPunc = 0
            fileappend 'outFile$' 'lineText$' 'newline$'
        # If it has any target symbol, keep tokenizing and print out.
        else
            while indexOfPunc <> 0
                sentence$ = left$(lineText$,indexOfPunc)
                fileappend 'outFile$' 'sentence$' 'newline$'
                lineText$ = right$(lineText$,(lengthOfLineText-indexOfPunc))
                lengthOfLineText = length(lineText$)
                # Remove any leading spaces if any.
                existSpace = startsWith(lineText$, " ")
                while existSpace = 1
```

```
                    lineText$ = right$(lineText$,(lengthOfLineText-1))
                    lengthOfLineText = length(lineText$)
                    existSpace = startsWith(lineText$, " ")
                endwhile
                # Then continue searching for the next sentence.
                indexOfPunc = index(lineText$,punc$)
                lengthOfLineText = length(lineText$)
            endwhile
            # Print the last token, if any, to the output text file.
            fileappend 'outFile$' 'lineText$' 'newline$'
        endif
    endfor
    select Strings outFileObj
    Remove
endproc
################################# END OF SCRIPT ################################
```

2. Chunk a long sound file by sentence

```
################################################################################
# chunkLongSoundBySentence.praat (Written by Kyuchul Yoon  kyoon@kyungnam.ac.kr)
################################################################################
# Before you run, check that you have the following ready.
# (1) A long sound (.wav) file.
# (2) A text file containing a list of sentences.
#     (Each line should have one sentence)
#-------------------------------------------------------------------------------
# This script will prompt the user with the long sound file and an annotation
# tier. The user should select, by clicking and dragging on the editor window,
# the sentences one by one sequentially as specified in the text file.
# When the user is done, the script will chunk sentences into smaller sound files
# whose filenames are preceded by arabic numerals. The starting number can be
# specified.
################################################################################
form Set parameters
    comment The following files should be in the same folder as this script.
    word longSoundFile_(wav) s0201a.wav
    word sentenceTierName sentences
```

```
    word sentenceListFile_(txt) sentenceList.txt
    comment Specify the starting number of the chunked smaller sound files.
    natural startingFileNum 20
    comment Specify the number of digits for the arabic numeral.
    natural numOfDigits_(minimum_of_2_to_maximum_of_5) 4
    comment Specify the folder (to be created) for chunked sound files. (Blank if same as long
    sound prefix)
    word outFolder
    comment Specify the prefix for the chunked smaller files. (Blank if same as long sound
    prefix)
    word prefixChunked
    comment Thus the chunked files will be named "prefix-001.wav/TextGrid"
    comment Specify the visible window length on an editor window
    real windowLength_(in_seconds) 30
    real startOfWindow_(in_seconds) 60
    comment A log file (.log) will be created with the long sound prefix.
endform

# Before creating the outFolder, check if it's left blank.
if length(outFolder$)=0
    outFolder$ = longSoundFile$ - ".wav"
endif
system_nocheck mkdir 'outFolder$'

# Check for the prefix of the chunked smaller sound/TextGrid files.
if length(prefixChunked$)=0
    prefixChunked$ = longSoundFile$ - ".wav"
endif

# Create the filename for the long textgrid.
prefix$ = longSoundFile$ - ".wav"
textgridName$ = prefix$ + ".TextGrid"

# Create the filename for the log file.
logName$ = prefix$ + ".log"
fileappend 'logName$' source 'tab$'start 'tab$'end 'tab$'sentence 'newline$'

# Display the contents of the text file in the info window.
```

```
Read Strings from raw text file... 'sentenceListFile$'
Rename... sentenceListObj
numSentences = Get number of strings
clearinfo
for iSentence to numSentences
    select Strings sentenceListObj
    # Store sentences in an array variable.
    lineText 'iSentence'$ = Get string... iSentence
    dummy$ = lineText'iSentence'$
    printline 'dummy$'
endfor
# Set the total number of sentences.
countSentence = iSentence

# Then open the long sound file in the editor window.
Open long sound file... 'longSoundFile$'
Rename... longSoundObj
To TextGrid... 'sentenceTierName$'
Rename... textgridObj

# Set the flag to zero. The flag will be used to see if the user has selected.
flag = 0
iSentence = 0

# Set the visible window size and location.
centerOfWindow = startOfWindow+(windowLength/2)
halfOfWindow = windowLength/2

while flag = 0
    select LongSound longSoundObj
    plus TextGrid textgridObj
    Edit
    editor TextGrid textgridObj
        # Set the visible window size
        farLeft = centerOfWindow-halfOfWindow
        farRight = centerOfWindow+halfOfWindow
        Zoom... farLeft farRight
        # Then pause for user input
```

```
    pause Select a sentence to annotate. DO NOT select to write.
    startOfSelection = Get start of selection
    endOfSelection = Get end of selection
    # Update the center of visible window to display for next time.
    centerOfWindow = startOfSelection+(endOfSelection-startOfSelection)/2
    # Check if the user has selected anything
    if (endOfSelection-startOfSelection) = 0
        flag = 1
        pause Write chunked sound files to 'outFolder$'/ folder?
    endif
    Close
endeditor
select TextGrid textgridObj
# Add boundaries only if the user has selected anything, i.e. flag=0
if flag = 0
    # Get the sentence
    iSentence = iSentence + 1
    if iSentence < countSentence
        sentence$ = lineText 'iSentence'$
        # Then insert boundaries.
        Insert boundary... 1 startOfSelection
        Insert boundary... 1 endOfSelection
        # Now, set the interval text.
        intervalNum = Get interval at time... 1 startOfSelection
        Set interval text... 1 intervalNum 'sentence$'
    else
        pause No more sentences in the text file! Write sound files to 'outFolder$'/ folder?
        flag = 1
    endif
endif
# Save the textgrid file for the long sound file.
select TextGrid textgridObj
Write to text file... 'textgridName$'

# Save the annotated chunks into smaller sound/textgrid files.
if flag = 1
    select TextGrid textgridObj
    numIntervals = Get number of intervals... 1
```

```
# Count the files.
fileCount = 0
for iInterval to numIntervals
    select TextGrid textgridObj
    intervalText$ = Get label of interval... 1 iInterval
    # If there's a sentence in the interval, then increase the file count.
    if length(intervalText$) <> 0
        fileCount = fileCount + 1
        # Get the start/end of the interval.
        startOfInterval = Get starting point... 1 iInterval
        endOfInterval = Get end point... 1 iInterval
        # Put the info in the log file.
        fileappend 'logName$' 'longSoundFile$'" 'ab$' 'startOfInterval:2'
        ...'tab$' 'endOfInterval:2' 'tab$' 'intervalText$' 'newline$'
        # And extract the segment.
        Extract part... startOfInterval endOfInterval no
        Rename... extractedPartTextGrid
        select LongSound longSoundObj
        Extract part... startOfInterval endOfInterval no
        Rename... extractedPartSound

        # Get the correct arabic numeral for the files.
        actualStartingFileNum = (startingFileNum-1)+fileCount
        if actualStartingFileNum < 10
            if numOfDigits = 5
                zeros$ = "0000"
            elsif numOfDigits = 4
                zeros$ = "000"
            elsif numOfDigits = 3
                zeros$ = "00"
            elsif numOfDigits = 2
                zeros$ = "0"
            endif
        elsif actualStartingFileNum < 100
            if numOfDigits = 5
                zeros$ = "000"
            elsif numOfDigits = 4
                zeros$ = "00"
```

```
            elsif numOfDigits = 3
                zeros$ = "0"
            elsif numOfDigits = 2
                zeros$ = ""
            endif
        elsif actualStartingFileNum < 1000
            if numOfDigits = 5
                zeros$ = "00"
            elsif numOfDigits = 4
                zeros$ = "0"
            elsif numOfDigits = 3
                zeros$ = ""
            endif
        elsif actualStartingFileNum < 10000
            if numOfDigits = 5
                zeros$ = "0"
            elsif numOfDigits = 4
                zeros$ = ""
            endif
        elsif actualStartingFileNum < 100000
            if numOfDigits = 5
                zeros$ = ""
            endif
        endif
        # Now save the files with the zeros.
        numeralPart$ = zeros$ + "'actualStartingFileNum'"
        prefixTemp$ = prefixChunked$ + "-"
        prefixFull$ = prefixTemp$ + numeralPart$
        soundNameChunked$ = prefixFull$ + ".wav"
        textgridNameChunked$ = prefixFull$ + ".TextGrid"
        select Sound extractedPartSound
        Write to WAV file... 'outFolder$'/ 'soundNameChunked$'
        Remove
        select TextGrid extractedPartTextGrid
        Write to text file... 'outFolder$'/ 'textgridNameChunked$'
        Remove
    endif
endfor
```

```
    endif
endwhile
pause Check the log file in the current folder!
select TextGrid textgridObj
plus LongSound longSoundObj
plus Strings sentenceListObj
Remove
################################## END OF SCRIPT ##################################
```

3. Create an annotation file, align by word and assign default labels

```
#############################################################################################
# createLabelAlignByEojeolAndAssignDefaultLabels.praat
#                                   (Written by Kyuchul Yoon  kyoon@kyungnam.ac.kr)
#############################################################################################
# Before you run, check that you have the following ready.
# (1) A subfolder containing chunked sound (.wav) and label (.TextGrid) files, prepared
#     by using "chunkLongSoundBySentence.praat".
#--------------------------------------------------------------------------------------------
# Here's what this script does.
# (1) Make a list of all the sound/textgrid files in the specified subfolder.
# (2) Prompt the user with the sound/textgrid editor window one at a time so that s/he
#     can align the space-delimited word in the word (or eojeol) tier (to be created).
# (3) Create the phonology tier with default phrase accent (LHa) and boundary tone (HL%)
#     labels. "LHa" stands for an Accentual Phrase and "HL%" stands for an Intonational
#     Phrase boundary. Each eojeol interval will be assigned default LHa labels and the
#     last eojeol (before the sentence period) a HL% label.
# (4) Create a break index tier with default indices: 2 for eojeol intervals and 3 for
#     the last eojeol.
# And after the aligning is done,
# (5) Assign default phonetics tier (to be created) labels for the aligned default
#     Accentual Phrases: for four or more syllable-eojeols, T+HL+Ha, for three-syllable
#     eojeols, T+LHa, for two-syllable eojeols, THa, and for one-syllable eojeol, Ta.
#     Depending on the identity of the first segment of each eojeol, the script will
#     assign a "H" for tense/aspirated obstruents, and a "L" otherwise.
# NOTE: The identity of the tense/aspirated consonants will depend on which romanization
#     scheme was used in the romanization of the hangul text.
# The script will also do the following when creating the eojeol tier.
```

```
# (1) Tokenize the sentence (from the "sentences" tier) by eojeol using spaces.
# (2) Each eojeol will have its own interval.
# (3) All the intervals will be placed on the right hand side of the tier depending on
#     the "factorOfDistance". If it's 2, the intervals will be on the right half of the
#     tier, if 3, on the right two thirds, and if 5, on the right four fifths.
#     The factor was added for different screen resolutions and for easier boundary
#     placement.
############################################################################################
form Set parameters
    comment Specify the input subfolder containing the sound/textgrid files.
    word inFolder s0201a
    word inSoundExt_(with_dot) .wav
    word inTextgridExt_(with_dot) .TextGrid
    comment Specify the output folder suffix. Aligned files will be moved to it.
    word outSuffix -aligned
    comment NOTE: Finished files will be moved to the output folder.
    comment Specify the syllable separator.
    word sylSep_(hyphen_for_default) -
    comment Specify new tier names.
    word wordTier_(interval_tier) word
    word phonologyTier_(point_tier) phonology
    word phoneticsTier_(point_tier) phonetics
    word breakIndexTier_(interval_tier) breakIndex
    word miscTier miscellaneous
    comment
    comment Specify the factor of distance. (Usually from 2 to 5)
    real factorOfDistance 3
    comment
    comment Specify the romanization scheme for the characters.
    optionmenu Romanization: 1
        option K-ToBI
        option Yale
        option McCune-Reischauer
        option Revised Romanization (Ministry of Culture 2000)
endform

# Create the output folder
outFolder$ = inFolder$ + outSuffix$
```

```
system_nocheck mkdir 'outFolder$'

# Make a list of all sound/textgrid files in the input folder and check for numbers.
Create Strings as file list... soundListObj 'inFolder$'/* 'inSoundExt$'
numSoundList = Get number of strings
Create Strings as file list... textgridListObj 'inFolder$'/* 'inTextgridExt$'
numTextgridList = Get number of strings
if numSoundList <> numTextgridList
    exit WARNING! The numbers of sound/textgrid files don't match!
endif

# Start looping through each pair of files.
for iFile to numSoundList
    select Strings soundListObj
    soundName$ = Get string... iFile
    Read from file... 'inFolder$'/ 'soundName$'
    Rename... soundObj
    prefix$ = soundName$ - inSoundExt$
    textgridName$ = prefix$ + inTextgridExt$
    Read from file... 'inFolder$'/ 'textgridName$'
    Rename... textgridObj

    # Work with the textgrid first. Tokenize the sentence by space.
    sentence$ = Get label of interval... 1 1
    indexOfSpace = index(sentence$, " ")
    lengthOfSentence = length(sentence$)
    i = 0
    while indexOfSpace <> 0
        i = i + 1
        word'i'$ = left$(sentence$,(indexOfSpace-1))
        sentence$ = right$(sentence$,(lengthOfSentence-indexOfSpace))
        indexOfSpace = index(sentence$, " ")
        lengthOfSentence = length(sentence$)
    endwhile
    # Store the last word
    i = i + 1
    word 'i'$ = sentence$
```

```
# Create tiers.
Insert interval tier... 2 'wordTier$'
# Insert <sil> intervals at the 1/20th of the beginning/end of the word tier.
totalDuration = Get total duration
silenceDurationAtBegin = totalDuration/20
silenceDurationAtEnd = totalDuration-silenceDurationAtBegin
Insert boundary... 2 silenceDurationAtBegin
Insert boundary... 2 silenceDurationAtEnd
Set interval text... 2 1 <silence>
Set interval text... 2 3 <silence>
# Now, insert the tokenized words with respect to the factor of distance.
# First, decide on the factored start and end between which the words will be assigned.
factoredStart = silenceDurationAtBegin+(silenceDurationAtEnd-silenceDurationAtBegin)/
factorOfDistance
factoredEnd = silenceDurationAtEnd
# Get the time interval for each word by dividing the factored length
# by the number of tokenized words minus one, because the first word will go
# to the non-factored side of the tier.
interWordInterval = (factoredEnd-factoredStart)/(i-1)
# Then, enter the words.
for j to i
    tokenizedWord$ = word'j'$
    Set interval text... 2 (j+1) 'tokenizedWord$'
    timeToInsertBoundary = factoredStart+interWordInterval*(j-1)
    if timeToInsertBoundary <> factoredEnd
        Insert boundary... 2 timeToInsertBoundary
    endif
endfor

# Prompt the user for aligning by the word.
plus Sound soundObj
Edit
pause Align by the word and press continue when done.

# Create additional tiers and assign default labels.
select TextGrid textgridObj
Insert point tier... 3 'phonologyTier$'
Insert point tier... 4 'phoneticsTier$'
```

```
Insert point tier... 5 'breakIndexTier$'
Insert interval tier... 6 'miscTier$'
numWordIntervals = Get number of intervals... 2
# We need to insert (numWordIntervals-2) points to the phonology tier.
numPoints = numWordIntervals-2
for k to numPoints
    timeToInsertPoint = Get end point... 2 (k+1)
    if k <> numPoints
        Insert point... 3 timeToInsertPoint LHa
    else
        # If it's the last point, then should be an IP boundary tone label.
        Insert point... 3 timeToInsertPoint HL%
    endif
endfor
# Now for the phonetics tier labels.
for m from 2 to (numWordIntervals-1)
    # Set the flag for aspirated/tense consonants.
    flagAspTns = 0
    intervalText$ = Get label of interval... 2 m
    # Get the start/end of the interval
    startOfInterval = Get starting point... 2 m
    endOfInterval = Get end point... 2 m
    lengthOfInterval = endOfInterval-startOfInterval
    # Check if the word starts with any of the aspirated/tense consonants.
    if romanization = 1
        call checkWordOnset 'intervalText$' 1
    elsif romanization = 2
        call checkWordOnset 'intervalText$' 2
    elsif romanization = 3
        call checkWordOnset 'intervalText$' 3
    elsif romanization = 4
        call checkWordOnset 'intervalText$' 4
    endif
    # Now, check the number of syllables for the current interval text.
    countSyllable = 0
    indexOfSylSep = index(intervalText$,sylSep$)
    lengthOfIntervalText = length(intervalText$)
    while indexOfSylSep <> 0
```

```
        countSyllable = countSyllable + 1
        intervalText$ = right$(intervalText$,(lengthOfIntervalText-indexOfSylSep))
        indexOfSylSep = index(intervalText$,sylSep$)
        lengthOfIntervalText = length(intervalText$)
    endwhile
    countSyllable = countSyllable + 1
    # We're ready to insert points to the phonetics tier.
    # Divide the cases into two: 5 or more syllables vs. up to 4 syllables.
    if countSyllable > 3
        for n to 4
            interPointInterval = lengthOfInterval/5
            timeToInsertPoint = startOfInterval+interPointInterval*n
            # Now enter the tone label.
            if (n=1 and flagAspTns=1)
                Insert point... 4 timeToInsertPoint H
            elsif (n=1 and flagAspTns=0)
                Insert point... 4 timeToInsertPoint L
            elsif n=2
                Insert point... 4 timeToInsertPoint +H
            elsif n=3
                Insert point... 4 timeToInsertPoint L+
            elsif n=4
                Insert point... 4 timeToInsertPoint Ha
            endif
        endfor
    elsif countSyllable=3
        for n to 3
            interPointInterval = lengthOfInterval/4
            timeToInsertPoint = startOfInterval+interPointInterval*n
            # Now enter the tone label.
            if (n=1 and flagAspTns=1)
                Insert point... 4 timeToInsertPoint H
            elsif (n=1 and flagAspTns=0)
                Insert point... 4 timeToInsertPoint L
            elsif n=2
                Insert point... 4 timeToInsertPoint +H
            elsif n=3
                Insert point... 4 timeToInsertPoint Ha
```

```
            endif
        endfor
    elsif countSyllable=2
        for n to 2
            interPointInterval = lengthOfInterval/3
            timeToInsertPoint = startOfInterval+interPointInterval*n
            # Now enter the tone label.
            if (n=1 and flagAspTns=1)
                Insert point... 4 timeToInsertPoint H
            elsif (n=1 and flagAspTns=0)
                Insert point... 4 timeToInsertPoint L
            elsif n=2
                Insert point... 4 timeToInsertPoint Ha
            endif
        endfor
    elsif countSyllable=1
            interPointInterval = lengthOfInterval/2
            timeToInsertPoint = startOfInterval+interPointInterval
            # Now enter the tone label.
            Insert point... 4 timeToInsertPoint Ha
    endif
endfor
# Break index tier. Need to insert (numWordIntervals-2) points to the break index tier.
for p to numPoints
    timeToInsertPoint = Get end point... 2 (p+1)
    if p <> numPoints
        Insert point... 5 timeToInsertPoint 2
    else
        # If it's the last point, then should be an IP boundary break index.
        Insert point... 5 timeToInsertPoint 3
    endif
endfor

pause Check files and press continue to move them to output folder.
# Save the aligned files to the output folder.
Write to text file... 'outFolder$'/ 'textgridName$'
Remove
filedelete 'inFolder$'/ 'textgridName$'
```

```
    select Sound soundObj
    Write to WAV file... 'outFolder$'/ 'soundName$'
    Remove
    filedelete 'inFolder$'/ 'soundName$'
endfor
select Strings soundListObj
plus Strings textgridListObj
Remove

procedure checkWordOnset word$ scheme
    if scheme = 1
        # K-ToBI romanization scheme.
        if (startsWith(word$, "p") or startsWith(word$, "t") or startsWith(word$, "k") or
        ... startsWith(word$, "P") or startsWith(word$, "T") or startsWith(word$, "K") or
        ... startsWith(word$, "c") or startsWith(word$, "C") or startsWith(word$, "s") or
        ... startsWith(word$, "S") or startsWith(word$, "h"))
            flagAspTns = 1
        endif
    elsif scheme = 2
        # Yale romanization scheme.
        if (startsWith(word$, "ph") or startsWith(word$, "th") or startsWith(word$, "kh") or
        ... startsWith(word$, "pp") or startsWith(word$, "tt") or startsWith(word$, "kk") or
        ... startsWith(word$, "ch") or startsWith(word$, "cc") or startsWith(word$, "s") or
        ... startsWith(word$, "ss") or startsWith(word$, "h"))
            flagAspTns = 1
        endif
    elsif scheme = 3
        # McCune-Reischauer romanization scheme.
        if (startsWith(word$, "p'") or startsWith(word$, "t'") or startsWith(word$, "k'") or
        ... startsWith(word$, "pp") or startsWith(word$, "tt") or startsWith(word$, "kk") or
        ... startsWith(word$, "ch'") or startsWith(word$, "tch") or startsWith(word$, "s") or
        ... startsWith(word$, "ss") or startsWith(word$, "h"))
            flagAspTns = 1
        endif
    elsif scheme = 4
        # Revised Romanization scheme.
        if (startsWith(word$, "p") or startsWith(word$, "t") or startsWith(word$, "k") or
        ... startsWith(word$, "pp") or startsWith(word$, "tt") or startsWith(word$, "kk") or
```

```
        ... startsWith(word$, "ch") or startsWith(word$, "jj") or startsWith(word$, "s") or
        ... startsWith(word$, "ss") or startsWith(word$, "h"))
            flagAspTns = 1
        endif
    endif
endproc
################################ END OF SCRIPT ################################
```

4. Do the annotation

```
#############################################################################
# doLabeling.praat (Written by Kyuchul Yoon  kyoon@kyungnam.ac.kr)
#############################################################################
# Before you run, check that you have the following ready.
# (1) A subfolder containing sound/textgrid files aligned by using
#     createLabelAlignByEojeolAndAssignDefaultLabels.praat
#-----------------------------------------------------------------------------
# Here's what this script does.
# (1) Opens a pair of aligned sound/textgrid files for K-ToBI labeling.
# (2) User does the labeling with the help of doLabeling.man Praat manpage.
# (3) When done, saves and moves the pair of files to the output folder.
#############################################################################
# IMPORTANT!!         Instructions on moving boundaries around       IMPORTANT!!
#############################################################################
# Please be sure to press the [SHIFT] key when dragging boundaries. This will
# ensure that other synchronized boundaries move simultaneously. Otherwise, you
# will destroy synchronization across other tiers.
#############################################################################
form Set parameters
    comment Specify input folder containing sound/textgrid files.
    word inFolder s0201a-aligned
    choice sound: 1
        button .wav
    choice textgrid: 1
        button .TextGrid
    comment Output folder will be created and will have "-labeled" suffix.
    comment Specify the main Praat manpage to open (Should be in the same folder as this
    script).
```

```
    word manFile 04-doLabeling.man
    comment Specify the script folder for the Praat manpage (Should be a subfolder)
    word manFolder manScripts
endform

# Create the output folder.
outFolder$ = inFolder$ + "-labeled"
system_nocheck mkdir 'outFolder$'

# Make a list of file names in the input folder.
Create Strings as file list... soundListObj 'inFolder$'/*.wav
numSounds = Get number of strings

# Run the Praat manpage.
Read from file... 'manFile$'

# Loop through each pair.
for iFile to numSounds
    select Strings soundListObj
    soundName$ = Get string... iFile
    prefix$ = soundName$ - ".wav"
    textgridName$ = prefix$ + ".TextGrid"
    Read from file... 'inFolder$'/ 'textgridName$'
    Rename... textgridObj
    Read from file... 'inFolder$'/ 'soundName$'
    Rename... soundObj
    plus TextGrid textgridObj
    Edit
    editor TextGrid textgridObj

    select TextGrid textgridObj
    pause Use the Praat manpage to do the labeling and press Continue when done.
    endeditor

    # Save the finished files to the output folder.
    select Sound soundObj
    Write to WAV file... 'outFolder$'/ 'soundName$'
    Remove
```

```
    filedelete 'inFolder$'/ 'soundName$'
    select TextGrid textgridObj
    Write to text file... 'outFolder$'/ 'textgridName$'
    Remove
    filedelete 'inFolder$'/ 'textgridName$'
endfor
select Strings soundListObj
Remove
################################## END OF SCRIPT ################################
```

ManPagesTextFile
"K-ToBI Labels" "Kyuchul Yoon" 20060423 0

<entry> "K-ToBI annotation labels and menu buttons."
<intro> "Select a point (in a point tier) or a range (by dragging on the spectrogram) and choose the following."
<intro> "To move boundaries around, please be sure to press [SHIFT] while dragging!"
<intro> "Be sure to use UNDO menu item under EDIT menu on the editor window when you make a mistake."
<entry> "%Point %tier %(i.e. %phonological %tone, %phonetic %tone, %break %index %tier) %menu"
<entry> "For phonological tone tier (#Tier #3)"
<list_item> "@@\SCmanScripts/man-phonology-delete.praat|delete@"
<list_item> ""
<list_item> "@@\SCmanScripts/man-phonology-LHa.praat|LHa@@@\SCmanScripts/man-phonology-HL%.praat|HL\% @@@\SCmanScripts/man-phonology-L%.praat|L\% @@@\SCmanScripts/man-phonology-H%.praat|H\% @@@\SCmanScripts/man-phonology-LH%.praat|LH\% @@@\SCmanScripts/man-phonology-LHL%.praat|LHL\% @@@\SCmanScripts/man-phonology-HLHL%.praat|HLHL\% @@@\SCmanScripts/man-phonology-HLH%.praat|HLH\% @@@\SCmanScripts/man-phonology-LHLH%.praat|LHLH\% @@@\SCmanScripts/man-phonology-LHLHL%.praat|LHLHL\% @"
<entry> "For phonetic tone tier (#Tier #4)"
<list_item> "@@\SCmanScripts/man-phonetics-delete.praat|delete@"
<list_item> ""
<list_item> "@@\SCmanScripts/man-phonetics-L.praat|L@@@\SCmanScripts/man-phonetics-H.praat|H@@@\SCmanScripts/man-phonetics-+H.praat|\+ H@@@\SCmanScripts/man-phonetics-L+.praat|L\+ @@@\SCmanScripts/man-phonetics-Ha.praat|Ha@@@\SCmanScripts/man-

phonetics-La.praat|La@"
<list_item> ""
<list_item> "@@\SCmanScripts/man-phonetics-X.praat|X@@@\SCmanScripts/man-phonetics-a.praat|a@@@\SCmanScripts/man-phonetics-%.praat|\% @"
<normal> "To add additional symbols"
<list_item> "@@\SCmanScripts/man-phonetics-ques.praat|\? @@@\SCmanScripts/man-phonetics-quesa.praat|\? a@@@\SCmanScripts/man-phonetics-ques%.praat|\? \% @"
<entry> "For break index tier (#Tier #5)"
<list_item> "@@\SCmanScripts/man-breakIndex-delete.praat|delete@"
<list_item> ""
<list_item> "@@\SCmanScripts/man-breakIndex-0.praat|0@@@\SCmanScripts/man-breakIndex-1.praat|1@@@\SCmanScripts/man-breakIndex-2.praat|2@@@\SCmanScripts/man-breakIndex-3.praat|3@@@\SCmanScripts/man-breakIndex-1m.praat|1m@@@\SCmanScripts/man-breakIndex-2m.praat|2m@@@\SCmanScripts/man-breakIndex-3m.praat|3m@@@\SCmanScripts/man-breakIndex-x.praat|x@ "
<entry> "%Interval %tier %(i.e. %miscellaneous %tier) %menu"
<entry> "For miscellaneous tier (#Tier #6)"
<list_item> "@@\SCmanScripts/man-misc-laughter.praat|\< laughter\> @"
<entry> "Manual Labeling Instructions."
<intro> "To delete a point : Select and press [ALT]-[Backspace] at the same time."
<intro> "To move around: Press [ALT]-[arrow keys]."
<intro> "To move boundaries around, please be sure to press [SHIFT] while dragging!"
<intro> "Be sure to use UNDO menu item under EDIT menu on the editor window when you make a mistake."

```
################################################################################
# man-phonology-delete.praat (Written by Kyuchul Yoon  kyoon@kyungnam.ac.kr)
################################################################################
# Here's what this script does:
# Within a Praat manpage, it will rename a point label in the phonology tier.
# This is just one of many scripts embedded in the manpage. Others were omitted
# to save space.
################################################################################
     cursor = Get cursor
endeditor

select TextGrid textgridObj
```

```
numPoints = Get number of points... 3
for i to numPoints
    timeOfPoint = Get time of point... 3 i
    if timeOfPoint = cursor
        Set point text... 3 i LHa
    endif
endfor
# Return to the original environment
editor
################################ END OF SCRIPT ################################
```

5. Evaluate inter-labeler agreement

```
#############################################################################
# rateInterLabelerAgreement.praat (Written by Kyuchul Yoon  kyoon@kyungnam.ac.kr)
#############################################################################
# Before you run, check that you have the following ready.
# (1) A subfolder containing a list of subfolders, each of which contains
#     the same set of TextGrid files used for inter-labeler agreement.
#  e.g.)  ..interLabelerAgreement/labeler01/
#                                 labeler02/
#                                 labeler03/
#                                 etc.
#     If there's a missing file in any of the subfolder, error message pops up.
#-----------------------------------------------------------------------------
# Here's what this script does.
# (1) Opens the same TextGrid file one at a time from each of the labeler folders.
#     e.g.) Opens "label_01.TextGrid" from all of the labeler folders.
# (2) Checks each phonological tone tier label (e.g. NONE, LHa, X%) corresponding
#      to the right boundary of each of the word tier interval.
# (3) Outputs labelers' labels to plain text files, one for each set of TextGrid.
# (4) Calculates various inter-labeler agreement rates for the phonological tone
#      tier. Does not calculate the surface (or phonetic) tone tier agreement.
#      The inter-labeler agreement numbers will be in each of the output text files.
# NOTE: Calculation follows Pitrelli et al. "Evaluation of prosodic transcription
#       labeling reliability in the ToBI framework", ICSLP 1994, pp.123-126.
#############################################################################
form Set parameters
```

```
    comment The following folder should have subfolders that correspond to the labelers.
    word inFolder interLabelerAgreement
    word inExt_(with_dot) .TextGrid
    natural tierOfWord 2
    natural tierOfPhonology 3
    comment A text report file (.agree) will be created for each TextGrid.
endform

# Make a list of subfolders. Each folder belongs to each labeler.
Create Strings as directory list... folderListObj 'inFolder$'
rawNumFolders = Get number of strings
# Since the raw number of folders contains self and parent, subtract it by 2.
numFolders = rawNumFolders-2

# Now, make a list of files for the first labeler folder.
firstFolderName$ = Get string... 3
Create Strings as file list... fileListObj 'inFolder$'/ 'firstFolderName$'/* 'inExt$'
Sort
numFiles = Get number of strings
pause 'numFolders' labelers & 'numFiles' files for each labeler. Continue?

# Get one folder name to initiate the loop
select Strings folderListObj
folderName$ = Get string... 3

# Initialize the numerical variables.
totalActualTPW = 0
totalReferenceTPW = 0

# Loop through each file in each folder
for iFile to numFiles
    select Strings fileListObj
    fileName$ = Get string... iFile
    Read from file... 'inFolder$'/ 'folderName$'/ 'fileName$'
    Rename... textgridObj
    numIntervals = Get number of intervals... tierOfWord
    # Noting the two silence intervals, get the number of eojeols.
    numOfEojeols = numIntervals - 2
```

```
Remove

# Get the output file name.
outFile$ = fileName$ + ".agree"

# String variables for the output file later on.
wordTierString$ = "WORD" + tab$
stringOfTPW$ = "ActualTPW" + tab$
stringOfRefTPW$ = "ReferenceTPW" + tab$

# Initialize the string variables for the output file later on.
for jFolder to numFolders
    stringOfLabeler'jFolder'$ = "labeler " + "'jFolder'"
    stringOfLabeler'jFolder'$ = stringOfLabeler 'jFolder'$ + tab$
endfor

# Initialize the numerical variables.
actualTPW = 0
referenceTPW = 0

# Loop through each interval to get the label in the phonology tier.
for iInterval from 2 to (numIntervals-1)
    # Get the interval text.
    select Strings folderListObj
    directoryName$ = Get string... 3
    Read from file... 'inFolder$'/ 'directoryName$'/ 'fileName$'
    Rename... textgridObj
    # Get the eojeol from the word tier and add it to the existing variable.
    textOfInterval$ = Get label of interval... tierOfWord iInterval
    wordTierString$ = wordTierString$ + textOfInterval$
    wordTierString$ = wordTierString$ + tab$
    Remove

    # Initialize the number of AP's or IP's.
    numOfIP = 0
    numOfAP = 0
    numOfNONE = 0
```

```
# Loop through each folder to get the TextGrid file.
for jFolder from 3 to rawNumFolders
    select Strings folderListObj
    directoryName$ = Get string... jFolder
    Read from file... 'inFolder$'/ 'directoryName$'/ 'fileName$'
    Rename... textgridObj
    # Get the end point of each interval.
    endPointOfInterval = Get end point... tierOfWord iInterval
    numPoints = Get number of points... tierOfPhonology

    # Look for the point label that corresponds to the end point of the interval.
    iPoint = 1
    flag = 0
    while (iPoint <= numPoints and flag = 0)
        timeOfPoint = Get time of point... tierOfPhonology iPoint
        # If there is a point label at the end point, then get the label.
        if timeOfPoint = endPointOfInterval
            flag = 1
            valueOfLabel$ = Get label of point... tierOfPhonology iPoint
        endif
        iPoint = iPoint + 1
    endwhile

    # If the flag is zero after the while loop, then there's no point label.
    if flag = 0
        valueOfLabel$ = "NONE"
    endif

    # Check the phonological tone tier point label for calculation.
    rightMostLetter$ = right$(valueOfLabel$,1)
    if rightMostLetter$ = "a"
        numOfAP = numOfAP + 1
    elsif rightMostLetter$ = "%"
        numOfIP = numOfIP + 1
    elsif rightMostLetter$ = "E"
        numOfNONE = numOfNONE + 1
    endif
```

```
            # Remove the object and store the label into an array variable
            # for the text file output later on.
            Remove
            iLabeler = jFolder - 2
            stringOfLabeler 'iLabeler'$ = stringOfLabeler 'iLabeler'$ + valueOfLabel$
            stringOfLabeler 'iLabeler'$ = stringOfLabeler 'iLabeler'$ + tab$
        endfor

        # Get the number of each label. If equal to or greater than 2, add to numOfTPW.
        totalNumOfTPW = 0
        if numOfNONE >= 2
            numOfTPW = numOfNONE*(numOfNONE-1)/2
            totalNumOfTPW = totalNumOfTPW + numOfTPW
        endif
        if numOfAP >= 2
            numOfTPW = numOfAP*(numOfAP-1)/2
            totalNumOfTPW = totalNumOfTPW + numOfTPW
        endif
        if numOfIP >= 2
            numOfTPW = numOfIP*(numOfIP-1)/2
            totalNumOfTPW = totalNumOfTPW + numOfTPW
        endif

        # Get the total number of transcriber-pair word (TPW) and add it to the existing
variable.
        numOfRefTPW = numFolders * (numFolders-1) / 2
        stringOfTPW$ = stringOfTPW$ + "'totalNumOfTPW'"
        stringOfTPW$ = stringOfTPW$ + tab$
        stringOfRefTPW$ = stringOfRefTPW$ + "'numOfRefTPW'"
        stringOfRefTPW$ = stringOfRefTPW$ + tab$

        # Get the total (accumulated) TPW.
        actualTPW = actualTPW + numOfTPW
        referenceTPW = referenceTPW + numOfRefTPW
    endfor

    totalActualTPW = totalActualTPW + actualTPW
    totalReferenceTPW = totalReferenceTPW + referenceTPW
```

```
    agreementRate = totalActualTPW / totalReferenceTPW * 100

    # Print the string output to a file.
    fileappend 'outFile$' 'wordTierString$' 'newline$'
    for jFile to numFolders
        dummy$ = stringOfLabeler 'jFile'$
        fileappend 'outFile$' 'dummy$' 'newline$'
    endfor
    fileappend 'outFile$' 'stringOfTPW$' 'tab$' 'actualTPW' 'newline$'
    fileappend 'outFile$' 'stringOfRefTPW$' 'tab$' 'referenceTPW' 'newline$'
    fileappend 'outFile$' 'newline$'
    fileappend 'outFile$' 'totalActualTPW' out of 'totalReferenceTPW' 'tab$'
    ... 'agreementRate:1'% 'newline$'
endfor
select Strings fileListObj
plus Strings folderListObj
Remove
################################ END OF SCRIPT ################################
```

6. Synchronize boundaries

```
#############################################################################
# synchronizeBoundary.praat (Written by Kyuchul Yoon  kyoon@kyungnam.ac.kr)
#############################################################################
# Before you run, check that you have the following ready.
# (1) A subfolder containing sound (.wav) and label (.TextGrid) files.
#-----------------------------------------------------------------------------
# Here's what this script does.
# (1) With respect to the reference tier, checks the target tier and displays a
#     list of "bad" boundaries that need synchronization along with their
#     difference in seconds. A reference constant (in seconds) would be required
#     to define the "bad" boundaries.
# (2) Based on the maximum difference, the script will actually adjusts the "bad"
#     boundaries and prompt the user to check the newly created target tier.
# (3) Removes the old target tier permanently and saves the new label file.
#############################################################################
form Set parameters
    comment SPECIFIY THE INPUT AND OUTPUT FILES AND FOLDERS.
```

```
    comment INPUT
    word inSound s0201a-aligned-labeled
    word inTextgrid s0201a-aligned-labeled-nonsync
    comment OUPUT
    word outSound_(blank_for_not_moving)
    word outTextgrid_(to_be_created) s0201a-aligned-labeled-sync
    comment _____________________________________________
    comment SPECIFY THE REFERENCE TIER AND THE TARGET TIER.
    comment The modified target tier will be created just below the target tier for your examination.
    comment After the examination, the old target tier will permanently be removed from the label file.
    natural referenceTier_(will_remain_the_same) 2
    optionmenu referenceTierType: 1
        option interval tier
        option point tier
    natural targetTier_(will_be_modified) 3
    optionmenu targetTierType: 2
        option interval tier
        option point tier
    comment _____________________________________________
    comment SPECIFY THE REFERENCE CONSTANT.
    comment Inter-tier boundary difference less than this value will be reported before actual modification.
    real referenceConstant 0.1
endform

# Check the outSound$ variable. If it's blank, the sound files stay put.
# Otherwise, create the output sound folder besides the output label folder.
outSoundLength = length(outSound$)
if outSoundLength <> 0
    system_nocheck mkdir 'outSound$'
endif
# Create the output label folder.
system_nocheck mkdir 'outTextgrid$'

# Check the tier types and divide the work.
if (referenceTierType = 1 and targetTierType = 2)
```

```
# This'd be the most common case, i.e. adjust the point tier wrt/
# the interval tier. E.g.) Adjusting the phonology/breakIndex tier wrt/
# the word tier.
# Read the files
Create Strings as file list... soundListObj 'inSound$'/*.wav
Sort
numFiles = Get number of strings
for iFile to numFiles
    select Strings soundListObj
    soundName$ = Get string... iFile
    prefix$ = soundName$ - ".wav"
    labelName$ = prefix$ + ".TextGrid"
    Read from file... 'inSound$'/ 'soundName$'
    Rename... soundObj
    Read from file... 'inTextgrid$'/ 'labelName$'
    Rename... labelObj

    # Loop through each point boundary on the target tier and compare
    # inter-tier boundary difference wrt/ the reference interval tier and
    # display them on the info window.
    select TextGrid labelObj
    numTargetPoints = Get number of points... targetTier
    numReferenceIntervals = Get number of intervals... referenceTier
    oldInterTierBoundaryDiff = 0
    maxValue = oldInterTierBoundaryDiff
    # Prepare the info window
    clearinfo
    printline indexOfPoint'tab$'interTierDiff
    for iPoint to numTargetPoints
        timeOfPoint = Get time of point... targetTier iPoint
        labelOfPoint$ = Get label of point... targetTier iPoint
        # Idenfity the closest boundary in the reference interval tier.
        flagFound = 0
        iInterval = 0
        while (flagFound = 0 and iInterval < numReferenceIntervals)
            iInterval=iInterval+1
            timeOfRtBoundary = Get end point... referenceTier iInterval
            interTierBoundaryDiff = abs(timeOfPoint-timeOfRtBoundary)
```

```
            # If the two inter-tier boundary difference is less than
            # the reference constant, then put it in the info window
            # and get the maximum value.
            if interTierBoundaryDiff < referenceConstant
                flagFound = 1
                printline 'iPoint' 'tab$''interTierBoundaryDiff'
                # Compare and set the new max value.
                if interTierBoundaryDiff > maxValue
                    maxValue = interTierBoundaryDiff
                    pointOfMaxValue = timeOfPoint
                    iPointOfMaxValue = iPoint
                endif
                # Now store everything to array variables.
                newTimeOfPoint 'iPoint' = timeOfRtBoundary
                newLabelOfPoint 'iPoint'$ = labelOfPoint$
            endif
        endwhile
    endfor
    # Now move the cursor near to the boundary where maxValue was calculated.
    # This is necessary so that the user can check if the reference constant
    # is appropriate. If not, run the script again with a different constant.
    select Sound soundObj
    plus TextGrid labelObj
    Edit
    editor TextGrid labelObj
        left = pointOfMaxValue-1
        right = pointOfMaxValue+1
        Select... left right
        Zoom to selection
        Move cursor to... 'pointOfMaxValue'
    endeditor

    # Comment the following 'pause' line for batch processing.
    # However, doing so might introduce errors.
    pause Max inter-tier boundary difference was 'maxValue:3' at
    ...'iPointOfMaxValue'th point. Perform adjustment?

    # Create a new point tier.
```

```
            select TextGrid labelObj
            oldTierName$ = Get tier name... targetTier
            newTierName$ = "new." + oldTierName$
            Insert point tier... targetTier 'newTierName$'
            for iPoint to numTargetPoints
                newTime = newTimeOfPoint'iPoint'
                newLabel$ = newLabelOfPoint'iPoint'$
                Insert point... targetTier newTime 'newLabel$'
            endfor

            # Comment the following 'pause' line for batch processing.
            # However, doing so might introduce errors.
            pause Check new/old target tier and Continue to remove the old tier and to save.

            oldTierNum = targetTier+1
            Remove tier... oldTierNum
            Set tier name... targetTier 'oldTierName$'
            if outSoundLength > 0
                Write to text file... 'outTextgrid$'/ 'abelName$'
                select Sound soundObj
                Write to WAV file... 'outSound$'/ 'soundName$'
            else
                Write to text file... 'outTextgrid$'/ 'labelName$'
            endif
            select Sound soundObj
            plus TextGrid labelObj
            Remove
        endfor
        select Strings soundListObj
        Remove

elsif (referenceTierType = 2 and targetTierType = 2)
        # This is also a possible scenario, e.g. adjusting the breakIndex tier wrt/
        # the phonology tier. This 2-2 case could be dealt with the 1-2 case above.
        exit Warning! Why don't you try referenceTierType(interval tier) and
        ...targetTierType(point tier)?

elsif (referenceTierType = 1 and targetTierType = 1)
```

```
    # This would be adjusting the misc tier wrt/ the word tier.
    # Would this be really necessary?
    # Could be coded later if need arises.
    exit Not coded yet!

elsif (referenceTierType = 2 and targetTierType = 1)
    # This'd be a pointless thing to do.
    # So, warn the user.
    exit Not coded yet!
endif
################################# END OF SCRIPT ################################
```

7. Normalize intensity of sound files

```
################################################################################
# normalizeIntensityOfSound.praat (Written by Kyuchul Yoon  kyoon@kyungnam.ac.kr)
################################################################################
# Before you run, check that you have the following ready.
# (1) A subfolder containing a list of sound files to be normalized
#-------------------------------------------------------------------------------
# Here's what this script does.
# (1) Scans the sound files to identify the maximum/minimum intensity in dB.
# (2) If not specified by the user, normalizes the sound files with respect to
#     the maximum value.
# (3) Outputs the normalized sound files in a different folder, preserving the
#     original sound files.
# NOTE: Uses the "Scale intensity..." menu item under "Modify -" button as of
#      version 4.4.13.
################################################################################
form Set parameters
    word inFolder orig-s0201a
    word inExt_(with_dot) .wav
    comment If blank, the prefix "normalized-" will be added to the input folder name.
    word outFolder_(to_be_created)
    comment If zero, the sound files will be scanned and normalized with respect to the
    maximum dB value.
    real valueOfdB 0
    comment If blank, the text log file will have the suffix ".log" after the input folder name.
```

```
    word logFile_(to_be_created)
endform

# Get the file/folder names.
if length(outFolder$) = 0
    outFolder$ = "normalized-" + inFolder$
endif
system_nocheck mkdir 'outFolder$'
if length(logFile$) = 0
    logFile$ = inFolder$ + ".log"
endif

# Check the number of sound files in the input folder.
Create Strings as file list... fileListObj 'inFolder$'/* 'inExt$'
numFiles = Get number of strings
pause 'numFiles' sound files identified. Continue?

# Check if the user specified the dB value to be used in the normalization process.
if valueOfdB = 0
    # Then scan the files to identify the maximum dB.
    pause No dB value specified. The maximum dB value will be used for normalization.
    maxdB = 0
    for jFile to numFiles
        select Strings fileListObj
        fileName$ = Get string... jFile
        Read from file... 'inFolder$'/ 'fileName$'
        Rename... soundObj
        dBvalue = Get intensity (dB)
        Remove
        # Establish the maximum value.
        if dBvalue > maxdB
            maxdB = dBvalue
        endif
    endfor
    # Now that we identified the maximum value, call the procedure.
    call procNormalize maxdB
else
    # Normalize the sound files with the user-specified dB.
```

```
    pause All the sound files will be normalized to 'valueOfdB' dB.
    call procNormalize valueOfdB
endif

procedure procNormalize value
    for iFile to numFiles
        select Strings fileListObj
        fileName$ = Get string... iFile
        Read from file... 'inFolder$'/ 'fileName$'
        Rename... soundObj
        beforeDB = Get intensity (dB)
        Scale intensity... value
        afterDB = Get intensity (dB)
        Write to WAV file... 'outFolder$'/ 'fileName$'
        Remove
        fileappend 'logFile$' 'fileName$' 'tab$'before: 'beforeDB:2' 'tab$'after: 'afterDB:2'
        'newline$'
    endfor
    select Strings fileListObj
    Remove
endproc
################## END OF SCRIPT #################
```

About the author:

윤규철(Kyuchul Yoon), 경남대학교 영어학부(Division of English, Kyungnam University)

449 Wolyong-dong, Masan, Kyungnam, 631-701, S. Korea

+82-55-249-2124

kyoon@kyungnam.ac.kr

〈국문초록〉

한국어 토비 표기를 위한 프랏 스크립트 툴

윤규철
(경남대 영어학부)[4]

이 논문의 목적은 한국어 토비 운율표기체제를 이용한 대용량의 음성말뭉치 개발 시에 인적 시간적 자원을 줄이기 위한 프랏 스크립트 툴을 소개하는 것이다. 여기에 소개된 프랏 스크립트는 연구자로 하여금 단순반복적인 단계를 생략할 수 있도록 도와주어 시간을 절약하도록 한다. 이 스크립트 세트를 이용하면 로마자화된 텍스트와 음성녹음을 토비체제로 표기된 음성말뭉치로 변환시킬 수 있다. 즉 텍스트와 음성녹음을 문장단위로 나누고, 또 표기자들 사이의 일치도를 평가하는 것에 이르기까지 일관된 작업을 반자동으로 수행할 수 있다. 소개된 스크립트는 원본공개프로그램의 규율을 따르며 필요에 따라 자유롭게 변환하여 이용할 수 있다.

* 열쇠글: 프랏, 스크립트, 한국어, 토비, K-토비, 운율표기

4) This work was supported by Kyungnam University Foundation Grant, 2006.