Internal Use Only

TR-I-0066

Modified MITalk

William J. Poser & Masanobu Abe

January 23, 1989

Abstract

This memorandum describes a modified version of the MITalk text-to-speech system that produces Japanese of a sort for experimental purposes. The modified version accepts romanized Japanese text as input and produces speech whose spectral properties are those of English but to which the phonological rules of English have not applied. The modified version allows the prosodic properties of the speech (duration and fundamental frequency) to be specified separately and merged with the segmental information generated by the higher levels of MITalk.

Copyright 1989 ATR Interpreting Telephony Research Laboratories
All Rights Reserved

CONTENTS

1. Introduction

The MITalk system in its normal form synthesizes English speech from textual input. This memorandum describes a modified version of MITalk that produces Japanese of a sort. The modified version accepts romanized Japanese text as input and produces speech whose spectral properties are those of English but to which the phonological rules of English have not applied. The modified version allows the prosodic properties of the speech (duration and fundamental frequency) to be specified separately and merged with the segmental information generated by the higher levels of MITalk.

2. The Organization of the MITalk System

MITalk consists of a top-level program which coordinates the execution of a sequence of modules.[1] The standard version of the program contains the following modules, which execute in this order:

FORMAT

Converts numbers written using Arabic numerals and special symbols such as %-sign and $-sign to words. Also handles pronunciation of abbreviations.

DECOMP

Attempts morphological analysis of input words based on search of lexicon. Produces part-of-speech information, morphological analysis, and phonemic representation of morphemes if possible.

PARSER

Produces a gross syntactic parse intended for the use of the prosodic rules.

SOUND1

Applies letter-to-sound rules to words not decomposed by **DECOMP**. Also applies stress rules and associated phonological rules, such as vowel reduction.

PHONO1

Applies prosodic rules. These seem to have to do mainly with destressing and marking of boundaries. However, there are four allophonic rules in this module as well.

PHONO2

Applies segmental phonological rules.

PROSOD

[1] Allen et al. (1987) provides an overall description of MITalk.

Applies durational rules. It also generates F0 information, but this is ignored by **F0TARG**.

F0TARG

Computes F0 targets. Produces an F0 value for the center of every segment and for the left boundary of every segment.

PHONET

Generates synthesizer control parameters at 5 ms. intervals. The F0 values are interpolated from the information supplied by **F0TARG**. The other parameters depend in part on table-lookup of specifications associated with each symbolic phone. Also takes care of coarticulatory effects.

VTCOEF

Converts control parameter specifications into a set of linear amplitude coefficients.

CWTRAN

Simulates the Klatt cascade/parallel formant synthesizer. It accepts 50 coefficient frames from **VTCOEF** and generates the next frame of the waveform.

This division into modules makes it fairly straightforward to replace an entire module or to break into the data stream and modify the data.

One other aspect of the system, namely the letter-to-sound rules, is easy to modify, since these rules are written in a special high-level rule specification language and are confined to a single file.

However, MITalk is not designed as a general development system for text-to-speech systems. With the exception of the letter-to-sound rules, the rules that it applies are hard-coded (with very few comments) and are designed specifically for English.

3. Modifications to the Standard Modules

The modifications to the standard modules eliminate the application of English phonological rules and orthographic conventions, and provide for the use of an input orthography appropriate for Japanese.[2]

(a) The module **DECOMP** has been modified so that it does not attempt dictionary lookup. Lookup of Japanese words in the English dictionary has the effect of bypassing the correct letter-to-sound

[2] If it were desired to use this system as a practical Japanese text-to-speech system rather than for the research purposes for which it was intended, it would be necessary to replace the **FORMAT** module as well in order to generate correct Japanese pronunciations of numerals and the like.

rules, so it is necessary either to avoid dictionary lookup or to replace the dictionary with one appropriate for Japanese.

(b) The module **SOUND1** has been modified so that it does not apply any phonological rules and so that it uses a revised set of letter-to-sound rules more appropriate for Japanese. The Japanese letter-to-sound rules are given in the Appendix.

(c) Four allophonic rules have been removed from the module **PHONO1**.

(d) The module **PHONO2** has been modified so that it does not apply any allophonic rules.

With these modifications, MITalk generates speech with English prosody and English spectral values for the phones, but without applying any English phonological rules or spelling conventions.[3]

4. Writing Letter-to-Sound Rules

One of the few parts of the **MITalk** system that is easy to modify are the letter-to-sound rules. However, the grammar of the letter-to-sound rules is not described in Allen et al. (1987) and there appears to be no other documentation available. Consequently, we describe here how to define new letter-to-sound rules. This description is based on reading the source code, studying the English letter-to-sound rules distributed with **MITalk**, and experimenting with the system. It is incomplete in that it describes only those aspects of the system that have been exercised. In particular, since we have had no use for prefix and suffix declarations and associated rules it contains little information on them.

4.1. Organization of the Letter-to-Sound Rule File

The letter-to-sound rules must be written in a single file. Lines beginning with a slash (/) are interpreted as comments; blank lines are ignored. All other lines are treated as meaningful by the rule compiler.

The file consists of seven sections, which must be ordered as follows:

prefixes	—	list of prefixes
suffixes	—	list of suffixes
variables	—	definitions of variables
consonant rules	—	rules applying to consonants
prefix rules	—	rules applying to prefixes
suffix rules	—	rules applying to suffixes
vowel rules	—	rules applying to vowels

Lines beginning with a percent-sign (%) denote the ends of sections. If the necessary section delimitters are omitted, the rule compiler will report

[3] In addition to the modifications related to removing English-specific aspects of the system, the modules **VTCOEF** and **CWTRAN** have been modified to use an output sampling rate of 12 KHz instead of the original 10 KHz.

an error and abort. If the section delimitters are present but no rules are given the rule compiler will not complain but the compilation of parts of the system will fail.

A skeletal letter-to-sound rule file is therefore:

```
/Prefix Declarations
%
/Suffix Declarations
%
/Variable Definitions
%
/Consonant Rules
%
/Prefix Rules
%
/Suffix Rules
%
/Vowel Rules
%
```

4.2. Prefixes and Suffixes

Prefixes and suffixes are morphemes to be treated specially. Even if you do not wish to declare any such morphemes, you must declare at least one prefix and at least one suffix due to a bug in the code. Failure to declare at least one prefix and at least one suffix causes the compilation of **sound1** to fail.

To get around this, you may declare prefixes and suffixes that do not occur in the language in which you are working. Note that prefixes and suffixes are represented using phoneme strings not as arbitrary letter strings.

4.3. Letter-to-Sound Rules

The letter-to-sound rules themselves are of the form:

letter_pattern > phoneme_pattern context_specifier left_context ____ right_context

where the context_specifier is one of],[,(, and). The phoneme_pattern must be a string of phoneme names, not arbitary characters. The phoneme names are not Klatt codes and do not correspond to normal phonetic practice, so some experimentation is necessary. Roughly speaking, single-letter codes

have their typical American English values, so you will be surprised if you follow phonetic practice or assume the values found in most languages written in the Roman alphabet. Related sounds, especially those usually written with digraphs in American English, are denoted by prefixing a circumflex. Thus, S represents the voiceless alveolar fricative, while ^S represents the voiceless palatal fricative.

The following table gives the correspondences between the phoneme symbols used in letter-to-sound rules and Klatt codes. The values of the Klatt codes may be found in Appendix B of Allen et al. (1987). Note that there is not a phoneme symbol for every Klatt code, so some rules that one might wish to write are not possible.

A	EY
B	BB
D	DD
E	IY
F	FF
G	GG
H	HH
I	AY
J	YY
K	KK
L	LL
M	MM
N	NN
O	OW
P	PP
R	RR
S	SS
T	TT
U	UW
V	VV
W	WW
Y	AX
Z	ZZ
^A	AA
^C	CH
^D	DH
^E	EH
^G	NG
^I	IH
^J	JJ
^L	EL
^M	EM
^N	EN
^O	AO
^S	SH
^T	TH
^U	UH
^Y	AH
^Z	ZH

```
"A    AE
"I    IX
"O    OY
"U    AW
```

The context_specifier indicates whether the contexts are phoneme contexts or letter contexts, as follows:

Specifier	Left Context	Right Context
(	phoneme	phoneme
)	letter	phoneme
[	phoneme	letter
]	letter	letter

The phoneme_pattern, left_context and right_context may be null; but the context_specifier and the >-sign may not be omitted. Note that the underscore in the context may not be omitted. That is, a rule that applies everywhere must have the form:

A > B context_specifier ___

A rule of the form:

A > B context_specifier

is ill-formed and will cause the rule compiler to abort after generating an error message indicating that it does not recognize the phoneme],[,(, or), whichever was given as the context specifier.

This system has no default rules. That is, it is necessary to provide a rule with no context to map each valid input symbol to itself, e.g.:

B > B] ___

which takes B to B everywhere.

However, it does observe the Elsewhere Principle, so it is not necessary to specify complementary environments. That is, if we want A to go to B before C and remain A elsewhere we need only write:

A > B] ___ C

A > A] ___

The first rule will apply before C, and the second rule in all other environments. It is not necessary to specify all environments other than before C in the second rule.

4.4. Variables

What MITalk refers to as variables are not really variables as they may not be assigned new values. Rather, they are constant sets. A set has a name that begins with a $-sign. The sets are defined by giving the name of the set followed by a comma and then the members of the set, separated by commas, e.g.:

$W,A,E,I,O,U,Y,+A,+E,+I,+O,+U,+Y,LE#,RE#

This line defines a set called $W to consist of A,E,I, etc., that is to say, roughly, the syllabic segments other than the nasals.

Sets are referred to in rules by their names, as in the following rule:

GU > GW] N ___ $W

This rule takes the sequence of letters GU to GW when preceded by the letter N and followed by any member of the set $W.

Note that in this language the $ is not a dereferencing operator as it is in some languages. It is simply part of the naming convention for identifiers. Thus, it is an error to attempt to define a set without using a $-sign in the name of the set. For example,

W,A,E,I,O,U,Y,+A,+E,+I,+O,+U,+Y,LE#,RE#

is not a legal set definition since the first identifier, "W", is not a legal set name.

The $-sign precedes the letter in the variable name in a variable definition and when the variable is mentioned to the right of the ___ in the environment of the rule. However, when the variable is mentioned to the left of the ___ in the environment of the rule, the $-sign and the letter are transposed. Thus, a rule that takes G to ^G between members of the set $B is written thus:

G > ^G] B$ ___ $B

Placing the $-sign on the left will cause **rulecomp** to abort with an unidentified phoneme error message.[4]

4.5. Installing the Letter-to-Sound Rules

MITalk expects to find the letter-to-sound rules in a file called **Source** in the directory $MITALK/src/sound1.[5] The file with this name in the

---

[4] This is not a joke.

[5] The environment variable MITALK is assumed to contain the name of the MITalk root directory. This directory is the parent of the MITalk src, lib, and bin directories. The original MITalk program had a bug that required the value of this variable to end in the slash that is used to separate path components. This bug has been fixed in the modified version.

MITalk distribution contains the standard rules for English; it is wise to save this file under another name before modifying or replacing it.

The new set of letter-to-sound rules should be put in a file called **Source** in the directory **$MITALK/src/sound1**. To compile the rules, **cd** to this directory and execute the program **rulecomp**. This compiles the rules into the form in which MITalk uses them and creates a number of files. You must then do a "make" to rebuild **sound1**, followed by a "make install" to install the modified version of **sound1** along with some of the files created by **rulecomp**.

5. Input to MITalk

Roughly speaking, this modified version of MITalk accepts input in the Hepburn romanization, with each utterance followed by a period. However, there are several details that it is necessary to consider.

First is the question of treatment of long segments, since there is no consonant length distinction in English and vowel length is associated with vowel quality. MITalk treats geminate consonants as single segments of greater duration than the corresponding singletons. Left to its own devices, MITalk treats double vowels as sequences of two segments. Since MITalk does nothing to produce the effect of rearticulation, these sound like a single long vowel. In order to simplify modification of the prosody, the Japanese letter-to-sound rules reduce clusters of like vowels to singletons, thereby neutralizing the contrast between long and short vowels. The appropriate duration can then be supplied at the prosody modification stage. In sum, this version of MITalk effectively ignores the distinction between long and short segments on the assumption that appropriate durations will be supplied separately.

Second is the problem of syllabification. MITalk provides no way to indicate syllabification directly. Where variants are associated with different positions in the syllable, these can be indicated by using special input symbols, provided that these can be mapped onto sounds known to MITalk. Thus, distinctions like that between /tani/ "valley" and /tan'i/ "unit" cannot be represented in terms of syllable structure. As a simple expedient the letter-to-sound rules have been written to allow the use of the letter *X*, which does not otherwise occur in romanized Japanese, to represent the mora nasal.[6]

Where the crucial distinction is one of syllabification pure and simple, MITalk provides no means of representing it at all. For example, MITalk cannot distinguish between *satooya* "sugar seller" [sato:ya], which has a single long [o], and *satooya* "foster parent" [sato$oya] which has a sequence of two short [o]s.[7]

---

[6] The symbol *N* commonly used for the mora nasal cannot be used here because the FORMAT module maps all alphabetic characters to upper case.

[7] A full text-to-speech system for Japanese would not only provide a representation for such distinctions but would generate them automatically, given morphological information, since potentially monosyllabic sequences of vocoids are normally hetero-

The inability to specify syllable structure also poses problems with *Cy* clusters. When such clusters are preceded by a vowel MITalk associates the initial consonant with the preceding syllable. Although this is not exactly what it should do, this at least yields output in which the consonant is produced. When no vowel precedes, MITalk effectively deletes the initial consonant since it has no provision for *Cy* clusters in syllable onsets.

For the purposes for which this version of MITalk was intended these limitations are not severe, but any attempt to use MITalk for general Japanese speech synthesis would require both a means of representing syllabification information and modification of the syllabification rules.

Finally, a few miscellaneous comments on the letter-to-sound rules are in order.

(a) No distinction is made between [ei] and [e:], both of which are treated the same as [e].

(b) /h/ is converted to /sh/ before /i/ whether or not the /i/ is voiceless, which correctly reflects the speech of many but not all Standard Japanese speakers.

(c) /g/ is realized as a nasal intervocalically and as an oral stop elsewhere. This reflects the speech of some speakers in some stylistic registers, but there is a great deal of variation in the context for nasalization of /g/.

6. Prosodic Information

The prosodic information generated by MITalk is specific to English, and depends upon the morphological and syntactic analysis performed by previous modules, which themselves are not easily modified. Consequently, instead of modifying MITalk itself, we have provided a means of replacing the prosodic information generated by MITalk with prosodic information from other sources.

6.1. How MITalk Handles Prosodic Information

MITalk generates durations in the module **PROSOD** and F0 values in the module **F0TARG**.[8] These values are then passed, together with the identity of the segment and a flag indicating whether or not the segment is in a stressed syllable, to the module **PHONET**, which interpolates the F0 values and generates parameter tracks for the formant synthesizer.

For each segment **F0TARG** generates two F0 target values. One is the target for the left boundary of the segment. The other is the target for the midpoint of the segment. **F0TARG** may generate non-zero values for F0 in

---

syllabic when separated by compound boundary and monosyllabic when tautomorphemic or when separated only by morpheme boundary.

[8] MITalk also generates F0 values in the module **PROSOD**, but these are ignored and overwritten by **F0TARG**.

voiceless segments. PHONET sets these to zero. F0TARG may generate negative values for F0. In these cases the real F0 value is the absolute value, and the sign is used to flag the presence of a glottal stop. This situation should not arise in this version of MITalk.

The F0 interpolation algorithm used by PHONET is linear interpolation through the sequence of left boundary targets and midpoint targets.Segmental perturbations of F0 are dealt with in F0TARG; PHONET does nothing but interpolation. The only exceptions to this are that:

(a) There is a special F0 pattern for glottal stop, which ramps F0 downward linearly from the midpoint target to 40 Hz over the last 50 ms. of the segment preceding the glottal stop.

(b) Voiceless regions have an F0 of 0.

6.2. Modifying Prosodic Information

The prosodic information generated by PROSOD and F0TARG may be modified by placing a filter between F0TARG and PHONET. The program **modify_prosody** is such a filter. It reads binary data in the format generated by F0TARG from the standard input, modifies the prosodic information, and writes the data on the standard output in the same format. The modified values are taken from files whose names are supplied on the command line. The input files should contain single precision (four byte) floating point numbers in binary format. The F0 values should be in Hz. **modify_prosody** will take care of the conversion to decihertz, which is the unit actually used by F0TARG and PHONET.

modify_prosody also sets a flag on every segment indicating that it is in a stressed syllable, since the result of cutting out the stress rules is that every syllable is regarded as unstressed. The reason that the stress flag is relevant in a system in which the allophonic rules are turned off and the prosodic information is to be replaced is that PHONET uses the stress flag to decide how much aspiration to generate for voiceless stops. Aspiration seems more natural with the stress flag set.

The parameter files should contain a leading value corresponding to the initial silence and a trailing value corresponding to the trailing silence. That is, for an N segment word they should contain N+2 values.

The parameter data file names are given as arguments to the option flags:

d — duration value

l — F0 target at left boundary of segment

m — F0 target at midpoint of segment

Therefore, to modify all three values the command line would look like this:

modify_prosody -d duration_file -l left_f0_file -m mid_f0_file

Any or all of these options may be omitted. If an option is omitted, the corrsponding parameter is not modified.

MITalk is normally run by executing the top-level program **mitalk** with command line arguments specifiying which modules are desired. **mitalk** sets up an appropriate pipeline, starts all of the necessary child processes, and passes necessary command-line arguments to them. In order to insert **modify_prosody** into the pipeline it is necessary to bypass **mitalk**. The following shell script may be used instead. A copy will be found, under the name **mod_mitalk**, in the MITalk bin directory. It sets up a pipeline including **modify_prosody** with the correct command-line arguments.[9] This script takes a single command-line argument, which is the name of the file containing the text to be synthesized. The script expects duration, left boundary F0, and midpoint F0 values to be in files with the suffixes **.d**, **.lf** and **.mf** respectively. If it is not desired to modify all three parameters, the unnecessary options to **modify_prosody** may be removed.

```
format < $1 | decomp $MITALK/lib/Lexicon | parser |
sound1 $MITALK/lib/ | phono1 | phono2 | prosod | f0targ |
modify_prosody -l $1.lf -m $1.mf -d $1.d |
phonet | vtcoef | cwtran > $1.sp
```

In order to execute this script succesfully, the user's path must contain the MITalk bin directory. The *csh* command

```
setenv path = ( $path $MITALK/bin)
```

adds the MITalk bin directory to the user's path.[10]

References

Allen, J., Hunnicutt, M.S., & D. Klatt (1987) *From Text to Speech: the MITalk system.* Cambridge: Cambridge University Press.

---

[9] Recall that the environment variable **MITALK** should contain the pathname of the MITalk root directory.

[10] The equivalent *sh* command is:

```
PATH=$PATH:$MITALK/bin
export PATH
```

Appendix — Japanese Letter-to-Sound Rules

These rules are taken from the file **$MITALK/src/sound1/Source**, with comments translated into English.

```
// Prefixes

// In order to evade a bug we define a prefix that cannot
// occur in Japanese input.

^DA

%

// Suffixes

// In order to evade a bug we define a suffix that cannot
// occur in Japanese input.

F 23 — ^DE

%

// Variables

/Vowels
$B,A,I,U,E,O

/Labials
$L,P,B,M

/Velars
$V,K,G

%

// Consonant Rules

// These two rules convert long consonants into the corresponding
// short consonants.

SSH   >   ^S   ]   __
TCH   >   ^C   ]   __
```

```
SH    >   ^S   ]  ___
CH    >   ^C   ]  ___
J     >   ^J   ]  ___

K     >   K    ]  ___
S     >   S    ]  ___
Z     >   Z    ]  ___
T     >   T    ]  ___
D     >   D    ]  ___
N     >   M    ]  ___ $L
N     >   ^G   ]  ___ $V
N     >   N    ]  ___
```

// /g/ becomes a velar nasal between vowels

```
G   >   ^G   ]  B$ ___ $B

G   >   G    ]  ___
X   >   ^G   ]  ___
```

// /h/ becomes /sh/ before /i/

```
H   >   ^S   ]  ___ I

H   >   H    ]  ___
F   >   F    ]  ___
P   >   P    ]  ___
B   >   B    ]  ___
M   >   M    ]  ___
Y   >   J    ]  ___
R   >   R    ]  ___
W   >   W    ]  ___
```

%

// Prefix Rules

// None

%

// Suffix Rules

// None

%

// Vowel Rules

// These rules convert long vowels into the corresponding short vowels.
// No distinction is made between [e:] and [ei].

```
AA   >   ^A   ]  ___
II   >   E    ]  ___
UU   >   U    ]  ___
EE   >   A    ]  ___
EI   >   A    ]  ___
OO   >   O    ]  ___

A    >   ^A   ]  ___
I    >   E    ]  ___
U    >   U    ]  ___
E    >   A    ]  ___
O    >   O    ]  ___

%
```