Multipurpose Large Vocabulary Continuous Speech Recognition Engine

Julius

rev. 3.2

(2001/12/03)

Translated from the original Julius-3.2-book by Ian Lane – Kyoto University

Julius

- Julius is a high performance continuous speech recognition software based on word N-grams. It is able to perform recognition at the sentence level with a vocabulary in the tens of thousands.
- Julius realizes high-speed speech recognition on a typical desktop PC. It performs at near real time and has a recognition rate of above 90% for a 20,000-word vocabulary dictation task.
- The best feature of the Julius system is that it is multipurpose. By recombining the pronunciation dictionary, language and acoustic models one is able to build various task specific systems. The Julius code also is open source so one should be able to recompile the system for other platforms or alter the code for one's specific needs.
- Platforms currently supported include Linux, Solaris and other versions of Unix, and Windows. There are two Windows versions, a Microsoft SAPI 5.0 compatible version and a Windows DLL version.
- This documentation relates to the Unix version of Julius. For documentation on the Windows versions see the Julius for SAPI README (CSRC CD-ROM) (Japanese), or the Julius for SAPI homepage (Kyoto University) (Japanese).

Contacts/Links

Contacts/Links

- Home Page: http://winnie.kuis.kyoto-u.ac.jp/pub/julius/ (Japanese)
- E-mail: julius@kuis.kyoto-u.ac.jp

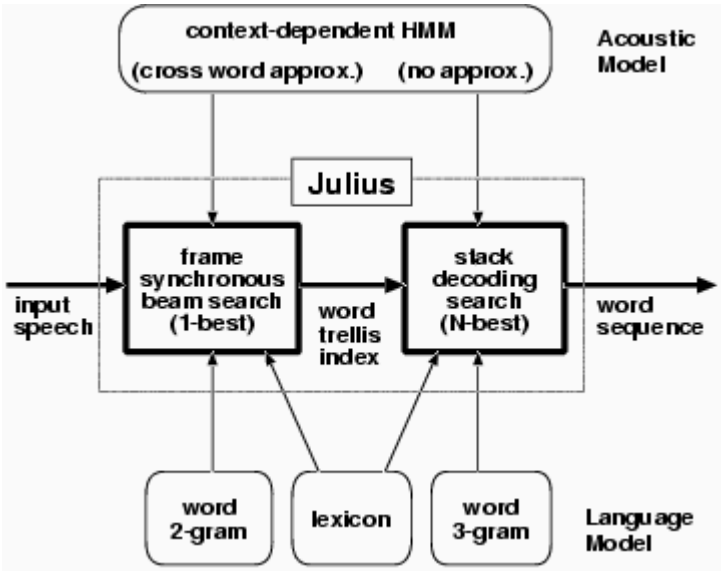
Developers

- Original System/Unix version: Akinobu Lee (ri@is.aist-nara.ac.jp)
- Windows Microsoft SAPI version: Takashi Sumiyoshi
- Windows (DLL version): Hideki Banno

System Structure and Features

System Structure

The structure of the Julius speech recognition system is shown in the diagram below:



- N-gram language models and a HMM acoustic model are used.
- The input speech is processed using a two-pass search.
  1. First Pass: 2-gram frame synchronous beam search. (High speed approximate search)
  2. Second Pass: 3-gram N-best stack decoding. (High Precision)

Main Features

- Can perform recognition with a vocabulary of up to 65,535 words.
- A recognition rate of over 95% for a 20k vocabulary dictation task using a high precision setting (5 times real time) and above 90% for real-time recognition.
- The system uses word 2-gram and (reverse) 3-gram language models. Either standard ARPA models or a Julius binary format model may be used. (A conversion tool is included)
- Monophone, triphone, or tied-mixture triphone models can be used. Julius uses HTK (Hidden Markov Toolkit) HMM definition files. Tied-mixture models are recognized by the joins within the tied-mixture codebook. Julius automatically detects the model type at startup.
- Online recognition can be performed using a PC microphone or a DatLink (NetAudio) device. Analysis automatically begins when speech is detected, and intermediate results can be shown in real-time. Julius can also use speech waveform files, and HTK format feature parameter files as input. When using file input, there is no limit on the file length.

Rev.3.2 New Features

Changes From Julius-3.1

All new features in version 3.2 are off by default. If you do not specifically set these options at run time then Julius-3.2 will execute in the same manner as version 3.1.

Recognition Algorithm Improvements

Sequential decoding using short-pause segmentation (Julius)

In the first pass the input is segmented by short pauses, the second pass sequential decodes these segments and slots the results in appropriately. In the first pass when a short pause (sp) has the maximum likelihood at a certain point in time, a break is placed at that point and the second pass is executed on that utterance segment. When this occurs, word constraints are preserved as the context within a utterance segment may continue over to the next utterance.

Using this feature one input file with multiple sentences can be decoded, thus it is not necessary to pre-process long speech files into sentence length utterances, files of any length can be used.

*At this time this option cannot be used with microphone input, only with input files (speech files, feature parameter files).

This option is by default off. Use the compile time option "--enable-sp-segment" to enable it.

High Speed Likelihood Calculation Using Gaussian Mixture Selection (Julius/Julian).

A method for the high-speed calculation of gaussian mixture likelihoods has been included. For each frame monophones likelihoods are calculated, and only for those with high likelihoods are triphones likelihoods calculated.

The default is off. The startup option "-gshmm" can be used to select the monophone model to use for mixture selection. "-gsnum" is used to set the number of mixtures to test (From all monophone states only the best N are selected). The monophone model used for GMS is created from a conventional monophone model using the attached tool mkgshmm.

Transparent word processing (Julius)

When performing N-gram calculations transparent word processing has been implemented. Specified transparent words can be skipped within a contextual frame. Transparent words can be defined within the recognition dictionary by placing "{}" rather then "[]" in the second column. These words will be treated as filler words.

Word insertion Penalty (Julian)

From Julian version 3.2 it is possible to include a word insertion penalty. For the first pass use the "-penalty1" option and use "-penalty2" for the second pass.

Extensions to Speech Input and Acoustic Analysis Section

Speech Input Segmentation (Julius/Julian)

- Files can undergo the same speech segmentation preprocessing that is used for microphone input. Use the startup option "-pausesegment".
- You can alter the segmentation parameters using the options below.
  - -lv: Input threshold level (0-32767)
  - -zc: Zero crossing threshold (Counts per second)
  - -headmargin: Length of silence at the start of the file (msec)
  - -tailmargin: Length of silence at the end of the file (msec)
- The speech input module uses threaded processing by default.
- For microphone input, if the input is less then 50 frames (0.5 sec) CMN will not be performed.

Acoustic Analysis (Julius/Julian)

- Sampling rates other then 16kHz may be used (Including microphone input).
- High and low pass filter cutoff frequencies can be set.
- Acoustic analysis parameters can be set.
  - -smpFreq: Sample frequency (Hz)
  - -smpPeriod: Sample Period (ns)
  - -fsize: Frame size (Number of samples)
  - -fshift: Frame shift (Number of Samples)
  - -hipass: High pass cutoff frequency (Hz)
  - -lopass: Low pass cutoff frequency (Hz)

Other

- To perform recognition on a number of files a filelist listing the names of the files can be used. Julius can then be run with the option "-filelist *filename*".
- In the case that there are errors within the recognition dictionary, the system can be forced to ignore these and perform recognition with the "-forcedict" option.
- For PTM/triphone models, it is no longer necessary to define monophones and biphones in the HMMList. For interword triphones when the relevant triphone are not defined in HMMList, the likelihood for all triphones that match the current context are calculated and those with the maximum likelihood are used.
- Various small bug-fixes, improvements in implementation (especially in the speech input and acoustic likelihood calculation area)

Platform

OS

The Unix version of Julius/Julian Rev.3.2 will run on the following platforms.

- PC Linux 2.2.18 with ALSA driver
- PC Linux 2.2.18 with OSS driver

Rev.3.1 was tested on the following platforms. Rev.3.2 should also run on these platforms with no problems.

- PC Linux 2.2.x
- FreeBSD 3.2-RELEASE
- Sun Solaris 2.5.1
- Sun Solaris 2.6
- SGI IRIX 6.3
- DEC Digital UNIX 3.2
- Digital UNIX 4.0
- Sun SunOS 4.1.3

For information on the Windows versions see the documents below.

- Julius for SAPI README (CSRC CD-ROM) (Japanese)
- Julius for SAPI homepage (Kyoto University) (Japanese)

Machine Spec

Min Spec: Pentium III 300MHz, 96MB memory
Recommended Spec: Pentium III 700MHz, 192MB memory

For microphone input a soundcard that can record at 16bits and the appropriate driver are required.

For more detailed information refer to "Microphone Input".

Installation

Here an explanation of the installation procedure for the Unix version of Julius is given. For the Windows version please refer to either the Julius for SAPI README (CSRC CD-ROM) (Japanese) or the Julius for SAPI homepage (Kyoto University) (Japanese).

Quick Start

Julius can be installed using the steps below.

```
% tar xzvf julius-3.2.tar.gz
% cd julius-3.2
% ./configure
% make
% make install
```

Each of the installation steps is explained below.

1. Preparation

By default Julius can only load RAW and WAV (no compression) audio files. However the libsndfile library allows Julius to load various other audio format files, including AIFF, AU, NIST, SND, and WAV(ADPCM) formats. At configuration time Julius automatically searches for the libsndfile library, which should be installed in advance.

```
 (If required)
Install the libsndfile library.
```

Next extract the Julius source package into the appropriate directory.

```
% gunzip -c -d julius-3.2.tar.gz | tar xvf -
% cd julius-3.2
```

2. *configure*

The configure script searches for information on the OS type, CPU endian, C compiler and required libraries, microphone support, and other information. It then automatically adapts the make file. First run the configuration script.

```
% ./configure
```

It is possible to set various options at configuration time. By default Julius is compiled with the optimal settings with respect to speed. Other settings can be used as shown below.

- To compile Julius for optimal precision:

```
% ./configure --enable-setup=standard
```

For a detailed explanation of configuration options refer to "Configure Options".

The default compiler is "gcc" and the default debug options are "-O2 -g". When using Linux the default debug options are "-O6 -fomit-frame-pointer". In other cases, the compiler settings can be changed by altering the appropriate environment variables, "CC" and "CFLAG", before running the configure script. The header path and other settings are set in the "CPPFLAGS" environment variable.

3. Compile

```
% make
```

Builds the julius, mkbingram, adinrec, adintool, and mkgshmm binary executable under the appropriate directories.

4. Install

```
% make install
```

Installs the binary executable in /usr/local/bin/ and the online manuals in /usr/local/man/.

This completes the installation.

Documentation

- Tutorial

  An explanation of the basic functioning of Julius.
- Microphone Input

  An explanation of microphone input. OS specific setup, input level control etc.
- Compile Time Options

  An in depth explanation of compile time options.
- File Types and Restrictions Format specifications of models and audio files supported by Julius.
- Triphones and HMMList An explanation of physical and logical triphones and the format of the HMMList file.

Tutorial -from initialization to recognition-

This tutorial describes the basic functioning of the Unix version of Julius. Here file preparation, system startup, and recognition are explained.

1. Preparation
2. Audio file recognition
3. Microphone Input recognition

1. Preparation

Note: Julius requires both acoustic and language models.

At execution time, at minimum the three files below are required.

Acoustic HMM definition file	HTKs' HMM format
Word dictionary file	Very Similar to the HTK dictionary format
Word N-gram language model	2-gram and reverse 3-gram models (Standard ARPA format), or binary N-gram format (Julius format)

When context dependent acoustic HMM models are used (triphone, PTM etc.) the following file is also required.

HMMList file	Julius format

For detailed information on each of the file types refer to File Formats and Restrictions.

Simple acoustic and language models have been included in the following distributions:

- Continuous Speech Recognition Consortium distributed CD-ROM
- IPA "Japanese Dictation Fundamental Software" distributed CD-ROM (Japanese)
- Ohmsha Textbook "Speech Recognition Systems" CD-ROM (Japanese)

This tutorial assumes that user is using one of the CD-ROM distributions above.

The Run-time Configuration File (jconf)

The model files, recognition parameters, and input source can be set in the "jconf" configuration file. The source archive contains a sample configuration file "Sample.jconf". Copy this file to the appropriate directory and edit it to fit your requirements.

As described above users who are using a CD-ROM distribution with acoustic and language models should use the "jconf" file provided on that CD-ROM.

2. Audio File Recognition

First recognition of audio files will be described.

It is assumed that the audio format is 16bit .WAV (no compression) or RAW (big endian)format. The sampling rate is dependent on the analysis conditions used to create the acoustic model, but generally this is 16kHz.

Execute Julius from the command line. Use the "-C" option to select the "jconf" file to be used.

```
% julius -C jconf_filename
```

Each of the option statements within the jconf file are treated as command line options. One can override these by placing an option after the "-C *jconf_filename*" setting. For example if we wish to use audio files as the system input and this has not been defined in the jconf file we can set this on the command line.

```
% julius -C jconf_filename -input rawfile
```

Once initialization has completed, the prompt below is displayed and the system waits for a response.

```
enter filename->
```

Enter a filename and Julius will perform recognition on that file.

The recognition process takes place in two passes. First 2-gram frame synchronous recognition is performed on the input. An example of the output of this first pass is shown below.

```
input speechfile: ./test.raw
97995 samples (6.12 sec.)
### speech analysis (waveform -> MFCC)
length: 610 frames (6.10 sec.)
attach MFCC_E_D_Z->MFCC_E_N_D_Z
### Recognition: 1st pass (LR beam with 2-gram)
.....
pass1_best:  SPECULATION IN TOKYO WAS AT THE N. COULD RISE BECAUSE
        OF THE REALIGNMENT
pass1_best_wordseq:  <s> SPECULATION IN TOKYO WAS AT THE N. COULD RISE
        BECAUSE OF THE REALIGNMENT </s>
pass1_best_phonemeseq: silB | s p E k y x l e S I n | I n | t o k y o l w a s | @ t | D i |
      E n | k U d | r Y z | b x k A z | A v I D i | r i x I Y n m I n t | silE
pass1_best_score: -14745.729492
```

pass1_best: First pass best hypothesis word sequence (interim result).

pass1_best_wordseq: Same as above but as a N-gram sequence.

pass1_best_phonemeseq: Best hypothesis phoneme sequence ("|" separates words)

pass1_best_score: The hypothesis score (Log-likelihood)

After the first pass finishes, the second pass is performed and a final recognition result is displayed. The 2nd pass uses the interim results from the first pass and searches these results using a 3-gram stack decoding technique.

```
### Recognition: 2nd pass (RL heuristic best-first with 3-gram)
samplenum=610
sentence1:  SPECULATION IN TOKYO WAS THAT THE YEN COULD RISE BECAUSE
        OF THE REALIGNMENT
wseq1:  <s> SPECULATION IN TOKYO WAS THAT THE YEN COULD RISE
        BECAUSE OF THE REALIGNMENT </s>
phseq1: silB | s p E k y x l e S I n | I n | t o k y o l w a s | D @ t | D i |
      y E n | k U d | r Y z | b x k A z | A v sp I D i | r i x I Y n m I n t | silE
score1: -14819.208008
30514 generated, 5681 pushed, 418 nodes popped in 571
```

sentence1: is the final recognition result. Once recognition is complete the system returns to the filename prompt.

The format of the recognition output can be altered. Using the "-progout" option at run time causes the first pass interim results to gradually be shown, while the "-quiet" option gives a simple output as shown below.

```
97995 samples (6.10 sec.)
pass1_best:  SPECULATION IN TOKYO WAS AT THE N. COULD RISE BECAUSE
      OF THE REALIGNMENT
sentence1:  SPECULATION IN TOKYO WAS THAT THE YEN COULD RISE BECAUSE
      OF THE REALIGNMENT
```

This completes the description of audio file recognition.

3. Microphone Input Recognition

Direct microphone input can also be recognized. Use the run-time option "-input mic" to use microphone input.

After initialization the prompt below will appear and the system will wait for input.

```
<<< please speak >>>
```

When one starts to speak into the microphone, first pass recognition processing will begin. When one stops speaking the system will switch to the second pass, the finalized recognition result will be displayed and the system will return to the above prompt.

Note that the first utterance after startup will not be recognized properly.

If recognition starts prematurely due to environmental noise or on the other hand, if you speak and the system does not begin recognition, then alter the microphone input level appropriately.

For more details on microphone usage refer to Microphone Input.

Microphone Input

Here, environment settings, restrictions, and other points of caution relating to microphone input with Julius/Julian-3.2 are explained.

1. Acoustic models for microphone input

At present the only possible feature extraction method that can take place within Julius/Julian is MFCC_E_D_NZ feature extraction (this is the same features that are used in IPA's acoustic models). All acoustic models that are to be used with Julius's direct microphone input must be in this format. When using acoustic models that use different feature formats, e.g. LPC, microphone input cannot be used. Care should be taken when dealing with other formats (including HTK parameter files).

2. Supported operating systems

At present the following OS are supported for direct microphone input.

1. Linux (kernel driver, OSS/Linux, ALSA)
2. Sun Solaris 2.x
3. Sun SunOS 4.x
4. SGI IRIX
5. FreeBSD

For each OS, details and warnings are as follows.

- Linux

On Linux platforms the following sound drivers are currently supported.

- Standard Kernel Drivers
- OSS/Linux 4Front Technologies Commercial Drivers
- Advanced Linux Sound Architecture

These are selected automatically at configuration time.

To specifically set the sound driver use the "--with-mictype=TYPE" option with *configure*. (TYPE can be set to "oss" or "alsa").

Capture at 16bit, 16kHz, mono-aural format is necessary.

Please note that at present whether speech data is captured adequately is largely dependent on the chipset and software driver being used. Commonly used "Sound-Blaster Pro compatible" soundcards have been found not to work with the Julius system. It has also been found that especially on laptop PC's that in many cases the quality of the sound cards 16bit capture capabilities is very poor. In such cases the Julius systems direct microphone input may not function adequately.

Under Linux Julius does not perform sound mixing internally, an external tool such as xmixer should be used to select the input device and set the input device volume.

Related Links:

- Linux Sound-HOWTO
- ALSA
- OSS/Linux

- Sun Solaris 2.x

Julius has been tested on Solaris 2.5.1 and 2.6 platforms.

The default device path is "/dev/audio". You can change the device path using the AUDIODEV variable before compiling Julius.

After startup has completed, the audio input device will be automatically switched to the microphone and the input volume will be set to 14.

- Sun SunOS 4.x

Julius has been tested with SunOS 4.1.3. It is necessary to use the <multimedia/*> headers at compile time.

The default device path is "/dev/audio". You can change the device path using the AUDIODEV variable before compiling Julius.

After startup has completed, the audio input device will be automatically switched to the microphone and the input volume will be set to 20.

- SGI IRIX

Julius has been tested with IRIX 6.3. (Julius will very likely run under system 5.x as well)

After startup has completed, the audio input will be automatically set to microphone input, however the input volume will not be set automatically. Use the 'apanel' command to set the volume manually.

- FreeBSD

Julius has been tested with FreeBSD Release 3.2 using the 'snd' driver. Use the compile option "--with-mictype=oss".

As audio mixing is not performed, an external tool must be used to select the input device (MIC/LINE) and set the input volume.

Sound card and driver problems are similar to those in the Linux system, see the Linux section above.

3. Compile Time Settings

It is not necessary to use any special configuration options at compile time to allow microphone input. The "configure" script automatically detects the OS type used and includes the necessary libraries. You should however check the final configure messages as shown below to make sure the appropriate sound driver was detected correctly.

```
mic API type      : oss (Open Sound System compatible)
```

In the case that automatic detection fails set the microphone driver type use "-with-mictype = TYPE", where TYPE is one of oss, alsa, freebsd, sol2, sun4, or irix.

From version 3,1p1 even if the driver is ALSA, OSS API is used by default. If one intents to use ALSA then they should use OSS emulation. Native ALSA API (beta version) can be used with the configuration option "--with-mictype = alsa".

4. Procedure for Microphone Input Recognition

Here microphone recognition with Julius is explained. In the case that an error occurs during startup please refer to sections 5 and 6 below.

Microphone input can be selected at startup using the "-input mic" option. If you do this then after initialization a prompt like that below will appear and the system will wait for a voice trigger. (Utterances that occur before the prompt appears will be disregarded).

```
<<< please speak >>>
```

After confirmation of the prompt, face the microphone and speak. The microphone should be held about 15 cm from the mouth and speech should be as clear as possible.

Once the input is greater then a certain level Julius begins the first pass processing. Analysis proceeds in parallel with the input. At the next long pause the first pass analysis is stopped, and the system then switches over to the second pass search. The system then outputs the final result, and waits for the next speech utterance. The process is then repeated.

! WARNING !
When using microphone input, The first utterance cannot be recognized properly. Real time processing uses the previous input to calculate the current CMN parameters, thus for the first input utterance CMN cannot be performed. For the first utterance say "Mic Test" or any arbitrary utterance, and that input will be used to update the CMN parameters. Normal recognition will be performed from the second utterance.

If the “-demo” option has been set at startup, then during the first pass real-time interim results will be displayed.

5. Recording volume and trigger level adjustment

When recognition is not performing adequately, it may be necessary to reset the volume or speech detection level.

When altering these settings it is best to first set the input volume and then reset the speech detection level. When the microphone input is not being detected properly first the microphone input level should be set. (It is often best to use an external sound/mixing tool to check that the volume is set correctly).

If sensitivity is low and voice detection is not occurring correctly, or on the other hand the voice detection trigger is too sensitive to external noise then the trigger level should be altered. The speech detection level is set at startup using the -lv option. The value has a range from 0 - 32767 (unsigned short). The default is 3000. Increase the speech detection level to decrease the sensitivity of the trigger, and decrease the level to increase the sensitivity of the trigger.

6. Verification of Voice Recording

From Rev.2.2 a program that captures a single utterance from the microphone, adinrec, is included. Using this, the quality of the captured speech can be checked.

```
% ./adinrec/adinrec myfile
```

When used as shown above, adinrec will record one utterance from the microphone to the file 'myfile'. As 'adinrec' uses the same voice capture routines as Julius, the quality of the recorded file will be the same as that used for recognition.

If no options are set, then the recorded file will be headerless 16kHz, monoral, signed 16bit big endian. It can be played back as shown below. (Using the sox or aplay utilities)

```
Linux/OSS: sox -t .raw -r 16000 -s -w -x -c 1 myfile -t ossdsp -s -w /dev/dsp
Linux/ALSA: aplay -f s16b -r -s 16000 myfile
Solaris2: sox -t .raw -r 16000 -s -w -c 1 myfile -t sunau -w -s /dev/audio
```

The recoded speech waveform file can be viewed using the tools below.

- spwave
- wavesurfer
- snd

Compile Time Options

Options that can be used with "configure" when compiling Julius-3.2 are given below.

Options to Modify Engine Settings

"--enable-setup=..." can be used to set the recognition algorithm to one of the three presets below.

1) standard	:	Standard	Highest precision, but low speed
2) fast	:	Fast	High speed, high precision (default)
3) v2.1	:	Version 2.1 compatible	All options OFF

You can see what the current Julius executable algorithm settings are by using the "-version" option at start up, as shown below.

```
% julius -version
###### booting up system
Julius rev.3.2 (fast)
built on jale at Sun Aug 12 20:58:44 JST 2001
 - compiler: gcc -O6 -fomit-frame-pointer
 - options: UNIGRAM_FACTORING LOWMEM2 PASS1_IWCD SCAN_BEAM
GPRUNE_DEFAULT_BEAM
```

Following is a more detailed explanation of the three setup types.

1) Standard --enable-setup=standard

In this setup precision is of particular importance. When triphone models are used, strict interword calculation is performed on the second pass. The error rate is decreased to less than 1%, but the 2nd pass processing time is dramatically increased. Also by default a safe pruning method is used for Gaussian pruning so not to introduce errors when using TM and PTM HMM models. (At run time other pruning technique can be selected by using the -gprune option)

2) Fast --enable-setup=fast (default)

In this setup the best balance between speed and precision is given. The gaussian beam pruning technique is used. No specified time limit is set.

3) Revision.2.1 compatible --enable-setup=v2.1

Here the same search algorithm that is used in Julius version 2.1 is used. 1-gram factoring is performed and 1 pass interword calculations are performed.

The differences in the three setup options above are shown in the following graph,
where an O indicates that the option is enabled.

	1-gram factoring	1st pass IWCD	2nd pass strict IWCD	tree separation	Gauss. pruning default method
--enable-	factor1	iwcd1	strict-iwcd2	lowmem2	
standard	O	O	O	X	safe
fast	O	O	X	O	beam
v2.1	X	X	X	X	safe

Setting First Pass Precision

By default Julius performs high-speed approximate calculations using 1-gram factoring and 1-best approximation. Thus the first-pass recognition contains approximation errors and the results of this pass are not optimal. (However these errors are usually recovered in the second pass)

In some cases only a 2-gram model may be available, or greater emphasis on the first pass precision may be required. In these cases the settings below can be used.

```
% ./configure --enable-setup=v2.1 --enable-iwcd1 --enable-wpair
```

With these settings, 2-gram factoring, interword triphone calculations and word-pair approximation rather then 1-best approximation is performed. With these changes the computational cost increases considerably however first pass precision is also gained.

Option Details

Details of individual *configure* options are given below.

* Compile and the installation environment.

--prefix=dir
Change the installation directory to 'dir'.
Executables are installed to ${dir}/bin
Manuals are installed to ${dir}/man/{man1,cat1}
The default is /usr/local/bin

--with-netaudio-dir=dir
When using DatLink, set the directory which contains the DatLink Netaudio library include and lib files.

* Microphone related (options)

--disable-pthread
Do not use a POSIX thread system for speech capture.

--with-mictype=TYPE
Specifies the microphone input device driver to be used
Select one of alsa, oss, sol2, sun4, and irix.
(default: automatic selection)

* Search algorithm options

--enable-sp-segment
Turn on successive decoding using short pause segmentation In the first pass where the input is segmented with short-pauses, the second pass is executed in parallel and the results are slotted in appropriately. Using this feature it is not necessary to preprocess long speech files into sentence length utterances. Files of any length can be used.

* At this time this option can only be used with file based input. Microphone input cannot use this option.

--enable-words-int
Change the word ID type from unsigned short to int. This increases the memory required by Julius, but enables a vocabulary of greater then 65535 words to be used. At

	present system performance when using this option is not guaranteed.
--enable-factor1	Perform 1-gram factoring in the first pass. When compared to the default (2-gram factoring), the processing speed is greatly increased, however first pass precision is lower.
--enable-lowmem2	Perform high frequency language tree separation. When using 1-gram factoring recognition precision drops if the width is too small.
--enable-iwcd1	Handle first pass interword context dependent triphones. The amount of processing required is increased, but recognition precision is also gained.
--enable-strict-iwcd2	Perform strict interword context dependent triphone search during the second pass. Little precision is gained over iwcd1. (Slight increase in precision.)
--disable-score-beam	In the second pass disable the score based beam search. This should usually be set to its default ON.
--enable-wpair	Use word-pair approximation in the first pass. Compared with the default 1-best approximation, the precision of the first pass will increase, however the calculation cost will also greatly increase.

File Types and Restrictions

The file formats that Julius can use are given below.
(All files can also be read from their compressed gzip (.gz) state)

1. HMM definition file	: HTK's HMM file format
2. HMMList file	: (Julius format)
3. Word N-Gram language model	: Standard N-gram ARPA or Julius binary format
4. Word dictionary file	: Similar to HTK's dictionary format
5. Microphone Input	: 16bit 16kHz Sampling
6. Sound files	: .wav, .raw, and other formats
7. Feature parameter files	: The same format as HTK's parameter format

Details of the usage and format of each file type are given below.

1. HMM definition file (Startup option: -h)

Julius can only load HMM definition files that are written in HTK's HMM definition language. The system automatically detects and loads the following three types of HMM acoustic models, monophone, triphone, and tied-mixture based HMMs.

When using triphone models, in order to convert the pronunciation in the dictionary to triphones, a HMMList file is required. (Described below). This lists the links between all tri-phones that may occur in the dictionary and the physical triphone models.

Julius does not implement all the HMM methods that are used in HTK, it only implements a subset. Special care needs to be taken especially with state transitions.

Format

(More detailed information on each item below is given in section 7.9 of "The HTK Book for HTK V2.0")

- Output distribution format
  Only continuous HMM models may be used, (vector based models can not be used). For phonetic Tied-Mixture models, mixture tying must be used.

Mixture Component weights and Stream weight vectors are optional. The variance vector is by default diagonal. InvCover, LLTCover, XForm, and FULL keywords cannot be used. Duration parameter vectors cannot be used.

- Conditional state transitions

  The entry and exit states have the following restrictions.

  - ✧ They cannot hold an output distribution
  - ✧ The entry state must have only one exit transition
  - ✧ The exit state must have only one entry transition

  For non-entry/exit states skips and loop are permitted.
- Sharing macros

  ~t(shared transition matrix), ~s(shared state distribution), ~m(shared Gaussian mixture component), ~v(shared diagonal variance vector) macros can be used. Other tying macros (~w ~u ~i ~x) cannot be used.
- Multiple input stream

  There can only be one input stream.

If the HMM definition file does not meet the restrictions above, Julius will output an error message and exit.

Size restrictions

There is no limit to the number of HMM definitions, conditional probabilities, or macros that can be used within one definition file. The system is only limited by memory size.

Tied Mixture model classification.

From version 3.0 tied-mixture based models are supported. As well as conventional tied-mixture models, using a single codebook, phonetic tied-mixture models with multiple phonetic codebooks may be used. As with HTK a codebook may have any number of definitions. However normal mixture distribution definitions cannot be used.

1. state-driven

   Calculation of monophone and shared transition triphones. Output likelihood calculations are performed by state, the cache also stores information by state.

2. mixture-driven

   Tied-mixture model calculations.

   Calculations are preformed by codebook (Gaussian distribution sets). For each frame, first the likelihood of all Gaussian distributions within the codebook are first calculated, and then all distributions likelihoods for each codebook are cached. HMM state output probabilities are calculated and weighted using the data in the cache of the corresponding codebooks.

The acoustic model is deemed to be a tied mixture model if the <Tmix> directive is present within hmmdefs. When Julius loads hmmdefs if the <Tmix> directive is present the Julius will treat the model as tied-mixture based, otherwise mixture-driven calculation methods will be used.

Due to this acoustic models must use the <Tmix> directive for Julius to perform tied-mixture calculations.

The <Tmix> directive is used in the same way as in HTK, refer to the corresponding section in “The HTK Book” for more details on the <Tmix> directive.

2. HMMList file (-hlist)

The HMMlist file is a dictionary of phoneme declarations of the form "e-i+q". It describes the mapping between the phonetic triphone and the actual *hmmdef* definition. This file is necessary when using triphone HMMs.

For information on the format of the HMMList file refer to Triphones and HMMList.

3. Word N-Gram language model

3.1 Standard ARPA format (-nlr, -nrl)

Julius uses 2-gram and reverse 3-gram Standard ARPA formats.

Format

When reading ARPA format files it is necessary to place the unknown language category (<UNK>) as the first entry of the 1-grams. (In the case that the model has been built using the CMU SLM Tool Kit the above restriction will be satisfied).

Within the word dictionary words without N-grams, will use the probability of the total unknown language category.

When 2-gram and reverse 3-gram models are loaded if a reverse 3-gram tuple does not match any 2-gram context then that tuple will not be used, a warning message will appear, and the language model will continue to load.

Size restrictions

The vocabulary is limited to 65535 words.

If the "-enable-word-int" option is used at compile time, then the limit will be extended to 2^31 words. It should be noted that at present there is no guarantee that Julius will work on all systems if this option is used.

When using this option caution needs to be taken, as it will be necessary to re-compile all binary N-grams, compiled under the original configuration

3.2 Binary N-gram format (-d)

Binary N-grams are created from 2-gram, and reverse 3-gram files (both in ARPA format). The utility "mkbingram" is used to do this.

By pre-compiling the language model, the startup time of the system is greatly reduced. Also the runtime size of the system is decreased.

Note that Julius's binary N-grams are not compatible with the CMU-TK binary N-gram format (.binlm)

4. Word dictionary files (-v)

Julius's word dictionary format is very similar to HTK's format. The difference is that the second field (Output Symbol) is not optional.

Format

- Field one (word name)
  It is necessary to use the same Japanese encoding system in the dictionary file and the word N-gram language files so the appropriate entries can be matched.
  Words that do not have N-gram entries are matched to the <UNK> N-gram. That N-gram probability will be that of the total corrected probabilities of all non-N-gram words.
- Field two (Output symbol)
  This is sent to the output queue as a recognition result. The value must be surrounded with brackets [ ]. If the symbol is [] then nothing will be outputted.
- Field three (HMM phoneme sequence)
  The HMM phoneme sequence is described using monophones. (When using triphones HMM's intraword context dependencies are automatically created when loading the dictionary)

[Example]

(It is not necessary for words to be sorted alphabetically.)

ABANDONMENT	[ABANDONMENT]	x b @ n d l n m l n t
ABBAS	[ABBAS]	@ b x s
ABBAS	[ABBAS]	@ b x z
ABBEY	[ABBEY]	@ b i
ABBOTT	[ABBOTT]	@ b x t
ABBOUND	[ABBOUND]	x b u d
ABIDE	[ABIDE]	x b Y d
ABILITIES	[ABILITIES]	x b l l l t i z
ABILITY	[ABILITY]	x b l l l t i
ABLAZE	[ABLAZE]	x b l e z

Size Limit

The recognition dictionary is limited to 65,535 words.

However, at configuration time if the "-enable-word-int" option is used the dictionary can be extended to 2^31 words. At present performance is not guaranteed when using this option.

5. Microphone Input (-input mic)

The microphone device must be able to sample at 16kHz,16bits.

The default maximum sample length is 20 seconds (320k samples). To increase this length increase the size of MAXSPEECHLEN in the include/sent/speech.h header file and re-compile the system.

At this point of time Julius can only extract MFCC_E_D_N_Z feature parameters internally. Care should be taken as acoustic models based on any feature extraction method other then MFCC_E_D_N_Z, can not be used with microphone input.

6. Speech waveform files (-input rawfile)

Speech waveform files must be in 16kHz,16bit format.

Julius automatically detects on loading the two file types given below,

1. RAW file (any file with no header information)
   16kHz, 16bit(signed short), mono, big-endian
2. Microsoft Windows WAV file
   16kHz, 16bit, no-compression

If the libsndfile library has been included at compile time then the file formats below may also be used for file input. (Refer to the libsndfile documentation for further information) All data must be in 16kHz, 16bit format. (Sound data cannot be resampled within Julius)

3. Microsoft WAV 16 bit integer PCM.
4. Apple/SGI AIFF and AIFC uncompressed 16bit integer PCM
5. Sun/NeXT AU/SND format (big endian 16bit PCM)
6. Dec AU format (little endian 16 bit PCM)
7. Microsoft IMA/DVI ADPCM WAV format (16 bits per sample compressed to 4 bits per sample).
8. Microsoft ADPCM WAV format (16 bits per sample compressed to 4 bits per sample)
9. Microsoft 8 bit A-law an u-law formats (16 bits per sample compressed to 8 bits per sample)
10. Ensoniq PARIS big and little endian, 16 bit PCM files (.PAF)

The libsndfile library can be downloaded from

http://www.zip.com.au/~erikd/libsndfile

Due to the nature of the Julius search algorithm, any extra long input will cause instability in the 2nd search pass. Thus it is desirable to punctuate sentences with short soundless pauses.

At this time Julius can only internally extract MFCC_E_D_N_Z feature parameters. Other feature based acoustic models cannot be used for speech waveform file recognition. An external parameter extraction tool must be used. This is described in more detail below.

7. Feature parameter files (-input mfcfile)

Julius can perform recognition of HTK style parameter files.

Format

Feature parameter formats (base, kind, qualifier) can be used for recognition. However it is necessary that the HMM acoustic model being used was trained using the same feature format. If any of the necessary parameters are not present then an error will occur.

If for some reason the HMM type checking does not function properly, you can use the -notypecheck option to turn off type checking.

Feature Parameter Types

It is necessary that the feature parameter format matches the original HMM training data feature format. However if all the necessary parameters for the HMM are held within the given feature parameter file, Julius will then automatically extract the appropriate parameters for recognition.

For example
If the parameter format below is used for training

MFCC_E_D_N_Z = MFCC(12)+ ΔMFCC(12)+ ΔPow(1) (CMN) 25-dimension

Then for recognition you can also use feature parameter files other then MFCC_E_D_N_Z, such as

MFCC_E_D_Z = MFCC(12)+Pow(1)+ΔMFCC(12)+ ΔPow(1)
(CMN) 26-dimension

or

MFCC_E_D_A_Z = MFCC(12)+Pow(1)+ ΔMFCC(12)+ ΔPow(1)
+ ΔΔMFCC(12) + ΔΔPow(1) (CMN) 39-dimension

The parameter file needs to contain all of the parameters used for the original training of the HMM model, extra data contained with in the file will not be used.

8. 1 Maximum hypothesis sentence length for one input

The default maximum hypothesis sentence length is 150 words.

In the case that an exceptionally long input sentence is inputted the error below will occur.

sentence length exceeded (> 150)

The maximum sentence length can be extended, by increasing the MAXSEQNUM definition in the include/sent/speech.h header file,

#define MAXSEQNUM 150

and then re-compile the system, as shown below.

% make distclean; make

Triphones and HMMList

This document describes how Julius handles context dependent phonemes models (triphones) and the HMMList file.

1. Context dependent phoneme models

When Julius is given a context dependent phoneme model (triphone), triphone declarations are created from the phoneme declarations in the recognition dictionary, and these are mapped to the corresponding HMMs.
To generate a triphone declaration from monophones, the phone "X" following a phone "L", and preceding a phone "R", the triphone is declared in the form "L-X+R". Below is an example of the conversion to a triphone declaration of the word "TRANSLATE".

```
TRANSLATE    [TRANSLATE]  t      r       @       n      s      l      e      t
                          |
TRANSLATE    [TRANSLATE]  t+r    t-r+@   r-@+n   @-n+s  n-s+l  s-l+e  l-e+t  e-t
```

In Julius phoneme declarations like this are created from the recognition dictionary and called "logical triphones". The actual HMM names defined in *hmmdefs* are called "physical triphones".

2. HMMList File

The mapping between the logical triphones and physical triphones are specified in the HMMList file. The HMMList file gives the mappings between all possible triphones and the HMM's that are defined in *hmmdefs*. Below are details of the format.

1. Only one mapping for a logical triphone can be made per line.
2. The first column contains the logical triphone, the 2nd column defines the corresponding HMM name in *hmmdefs*.
3. If the triphone uses the same name as that defined within *hmmdefs* then the 2nd column is empty.
4. All logical triphones that may occur must be defined.
5. If a triphone is mapped to itself then an error will occur.

Below is an example. Entries that have an empty 2nd column show that the triphone name relates to an HMM that is directly defined within *hmmdefs*.

```
a-k
a-k+a
a-k+a: a-k+a
a-k+e
a-k+e: a-k+e
a-k+i
a-k+i: a-k+i
a-k+o
a-k+o: a-k+o
a-k+u
a-k+u: a-k+u
...
```

The actual mapping the system uses can be checked using the Julius run-time option "-check ". After finishing initialization an input prompt will appear, here enter the logical HMM triphone name and information relating to that triphone will be displayed.

3. Dealing with Inter-word Triphones.

For Julius the type of phoneme context used is different for the two passes. On the first pass, from the intra-word triphones that match the current context, those with the maximum likelihoods are used. For example, for the end boundary of the word "TRANSLATE" the likelihoods of the HMMs "e-t+a", "e-t+e", "e-tn+u" etc are calculated, and those with the maximum likelihood are assigned.

In the second pass more precise inter-word contexts are calculated. When the following word is expanded from a hypothesis, the search for the next model takes context into consideration.

```
TODAY + IS      t+x      t-x+d    x-d+e    d-e        l+z     l-z
          |
TODAY IS        t+x      t-x+d    x-d+e    d-e+l    e-l+z     l-z
```

4. Warnings When Creating HMMList Files

Take care of the following points when creating HMMList files.

The assignments in the HMMList file overwrite the HMM definitions in hmmdefs. In other words, if a definition name within hmmdefs is the same as one used in a mapping in HMMList, the mapping has more priority. For example if hmmdefs contains the definition:

```
~h "s-l+z"
```

And HMMList contains the mapping:

```
s-l+z z-l+z
```

then the HMM "s-l+z" will not be used, instead "z-l+z" will be used.

Man Pages

Online manuals:

- julius: Large Vocabulary Continuous Speech Recognition Engine
- julian: Continuous Speech Recognition Grammar Based Parser
- adinrec: Microphone Input Recording Tool
- mkbingram: Binary N-gram Compilation Tool
- mkgshmm: GMS Monophone Conversion Tool

NAME

Julius - Japanese LVCSR engine

SYNOPSIS

julius [-C jconffile] [options ...]

DESCRIPTION

Julius is a open source speech recognition engine that can perform continuous speech recognition with a vocabulary in the tens of thousands of words. High precision recognition can be obtained using a 3-gram based two pass search technique.

Julius can perform recognition on microphone input, audio files, and feature parameter files. Also as standard format acoustic models and language models can be used, these models can be changed to perform recognition under various conditions.

The maximum vocabulary is 65,535 words.

Model Usage

Julius uses the following models.

Acoustic Models

Acoustic HMM(Hidden Markov Model) are used. Phoneme models (monophone), context dependent phoneme models (triphone), tied-mixture and phonetic tied-mixture models can be used. When using context dependent models, interword context is taken into consideration. Files written in HTKs HMM definition language can be used.

Language Model

The system uses 2-gram and reverse 3-gram language models. Standard format ARPA files are used. Binary format N-gram models built using the attached tool mkbingram can also be used.

Speech Input

It is possible to recognize live input from either a Microphone, A-D or a DatLink (NetAudio) system. Speech waveform files (16bit WAV (no compression), or RAW format) and feature parameter files (HTK format) can be used.

Warning: Julius can only extract MFCC_E_D_N_Z features internally. If it is necessary to use HMMs based on another type of feature extraction then microphone input and speech waveform files cannot be used. Use an external tool such as wav2mfcc to create the appropriate feature parameter files.

Search Algorithm

Julius recognition is based on a two pass strategy. On the first pass the entire input is process and a interim result is displayed. The model used in this pass is a word 2-gram and a word HMM tree structured network. Decoding is performed by a frame synchronous beam search.

The second pass search using a reverse 3-gram model, this attempts to gain a higher precision recognition result. Word unit stack decoding is performed using the restrictions from interim results of the first pass and forward information.

When using context dependent phones (triphones), interword contexts are taken into consideration. For tied-mixture and phonetic tied-mixture models, high-speed acoustic likelihood calculation is possible using gaussian pruning.

OPTIONS

The options below allow you to select the models used and set system parameters. You can set these option at the command line, however it is recommended that you combine these options in the jconf settings file and use the "-C" option at run time.

Below is an explanation of all the possible options.

Speech Input

-input {rawfile|mfcfile|mic|netaudio|adinserv}

Select the speech wave data input source.
(default: mfcfile)
For information on file formats refer to the Julius documentation.

-NA server:unit

When using (-input netaudio) set the server name and unit ID of the Datlink unit to connect to.

-filelist file

With (-input rawfile|mfcfile) perform recognition on all files contained within the target filelist.

-adport portnum

With (input adinserv) A-D server port number.

Speech segmentation

-pausesegment

-nopausesegment

Force speech segmentation (segment detection) ON / OFF.
(For mic/adinnet default = ON. For files, default = OFF)

-lv threslevel

Amplitude threshold (0 - 32767). If the amplitude passes this threshold it is considered to be the beginning of a speech segment, if it drops below this level then it is the end of the speech segment.
(default: 3000)

-headmargin msec

Margin at the start of the speech segment (msec).
(default: 300)

-tailmargin msec

Margin at the end of the speech segment (msec).
(default: 400)

-zc zerocrossnum

Zero crossing threshold. (default: 60)

-nostrip

Depending on the sound device, invalid "0" samples
at the start and end of recording may not be removed
automatically. The default is to perform automatic removal.

Acoustic Analysis

-smpFreq frequency

Sampling frequency (Hz).
(default: 16kHz = 625ns)

-smpPeriod period

Sampling rate (ns)
(default: 625ns = 16kHz)

-fsize sample

Analysis window size (No. samples).
(default: 400, 25mS)

-fshift sample

Frame shift (No. samples). (default: 160, 10mS)

-hipass frequency

Highpass filter cutoff frequency (Hz).
(default: -1 = disable)

-lopass frequency

Lowpass filter cutoff frequency (Hz)
(default: -1 = disable)

Language Model(N-gram)

-nlr 2gram_filename

2-gram language model filename (standard ARPA format)

-nrl rev_3gram_filename

Reverse 3-gram language model filename. This is
required for the second search pass. If this is
not defined then only the first pass will take
place.

-d bingram_filename

Use a binary language model as built using mkbingram(1). This is used in place of the "-nlr" and "-nlr" options above, and allows Julius to perform rapid initialization.

-lmp lm_weight lm_penalty

-lmp2 lm_weight2 lm_penalty2

Language model score weights and word insertion penalties for the first and second passes respectively.

The hypothesis language scores are scaled as shown below

lm_score1 = lm_weight * 2-gram_score + lm_penalty

lm_score2 = lm_weight2 * 3-gram_score + lm_penalty2

The actual hypothesis word score is a N-gram log-likelihood which is scaled using the appropriate factors given below.

The default values are dependent on the language model:

First-Pass	Second-Pass	
5.0/-1.0	6.0/0.0	(monophone)
8.0/-2.0	8.0/-2.0	(triphone,PTM)
9.0/8.0	11.0/-2.0	(triphone,PTM,engine=v2.1)

-transp float

Insertion penalty for [transparent words]. (default: 0.0)

Word Dictionary

-v dictionary_file

Word Dictionary File (Required)

-silhead {WORD|WORD[OUTSYM]|#num}

-siltail {WORD|WORD[OUTSYM]|#num}

Sentence start and end silence as defined in the word dictionary.
(default: "<s>" / "</s>")

These are dealt with specially during recognition to hypotheses start and end points (margins). They can be defined as shown below.

	Example
Word_name	<s>
Word_name[output_symbol]	<s>[silB]
#Word_ID	#14

(Word_ID is the word position in the dictionary file starting from 0)

-forcedict

Disregard dictionary errors.
(Skip word definitions with errors)

Acoustic Model(HMM)

-h hmmfilename

The name of the HMM definition file to use.
(Required)

-hlist HMMlistfilename

HMMList filename. Required when using triphone based HMMS. Details are contained in the Julius documentation.
This file provides a mapping between the logical triphones names genertated from the phonetic representation in the dictionary and the HMM definition names.

-force_ccd / -no_ccd

When using a triphone acoustic model these options control interword context dependency. If neither of

these options are set then the use of interword
context dependency will be determined from the
models definition names.
If the "-force_ccd" option is set when using
something other then a triphone model, there is no
guarantee that Julius will run.

-notypecheck

Do not check the input parameter type.
(default: Perform the check)

-iwcd1 {max|avg}

When using a triphone acoustic model set the
interword acoustic likelihood calculation method
used in the first pass.

max: The maximum identical context triphone value (default)

avg: The average identical context triphone value

Options for tied-mixture and PTM acoustic models

-tmix K

When performing gaussian pruning, only calculate the upper
k gaussian densities per codebook. (default: 2)

-gprune {safe|heuristic|beam|none}

Set the gaussian pruning technique to use.
(default: safe (standard) beam (high-speed))

-gshmm hmmdefs

Set the Gaussian Mixture Selection monophone acoustic
model to use. A GMS monophone model is generated
from an ordinary monophone HMM model using the
attached program mkgshmm(1).
(default : none (do not use GMS))

-gsnum N

When using GMS, only perform triphone calculations
for the top N monophone states. (default: 24)

Short pause segmentation

-spdur

Set the sp threshold length for use in the first pass (number of frames). If number of frames that the sp "unit" has the maximum likelihood is greater then this threshold then, interrupt the first pass and start the second pass. (default: 10)

By default short pause segmentation is not used. At configuration time the "--enable-sp-segment" option must be used to perform segmentation.
(For details refer to the Julius documentation)

Search Parameters (First Pass)

-b beam_width

Beam width (Number of HMM nodes).
As this value increases the precision also increases, however, processing time and memory usage also increase.

default values: Model dependent,
400 (monophone)
800 (triphone,PTM)
1000 (triphone,PTM,engine=v2.1)

-sepnum N

(Used with the configure option "--enable-lowmem2")
Number of high frequency words to separate from the dictionary tree. (default: 150)

-1pass

Only perform the first pass search. This mode is automatically set when no 3-gram language model has been specified (-nlr).

-realtime

-norealtime

Explicitly state whether real time processing will be used in the first pass or not. For file input the default is OFF (-norealtime), for microphone, or NetAudio network input the default is ON (-realtime). This option relates to the way CMN is performed: when OFF CMN is calculated for each input independently, when the realtime option is ON the previous 5 second of input is always used. Refer to -progout.

Search Parameters (Second Pass)

-b2 hyponum

Hypothesis envelope width. This number of hypotheses are expanded (sorted by length), shorter hypothesis are not expanded. This prevents search failures. (default: 30)

-n candidate_num

The search continues until "candidate_num" sentence hypothesis have been found. These hypotheses are re-sorted by score and the final result is displayed. (Refer to the "-output" option). As Julius does not strictly guarantee a optimal second pass search, the maximum likelihood candidate is not always given first.

As this value is increased the probability that the maximum likelihood hypothesis is returned increases, but as a prolonged search must be performed, the processing time also becomes large. (default: 1)

default value is dependent on the recognition engine settings ("--enable-setup= ").

10 (standard)

1 (fast,v2.1)

-output N

Used with the "-n" option above. Output the top N sentence hypothesis. (default: 1)

-sb score

Score envelope width. For each frame, do not scan areas that deviate from the highest score by more then this envelope. This directly relates to the speed of the second pass acoustic likelihood calculations. (default: 80.0)

-s stack_size

The maximum number of hypothesis that can be stored on the stack during the search. A larger value gives more stable results, but increases the amount of memory required. (default: 500)

-m overflow_pop_times

Number of expanded hypotheses required to discontinue the search. If the number of expanded hypotheses is greater then this threshold then, the search is discontinued at that point. The larger this value is, the longer the search will continue, but processing time for search failures will also increase. (default: 2000)

-lookuprange nframe

When performing word expansion, this option sets the number of frames before and after in which to consider word expansion. This prevents the omission of short words but, with a large value, the number of expanded hypotheses increases and system becomes slow. (default: 5)

Forced alignment

-walign

Return the result of viterbi alignment of the word units from the recognition results.

-palign

Return the result of viterbi alignment of the phoneme units from the recognition results.

Message Output

-separatescore

Output the language acoustic scores separately

-quiet Omit phoneme sequence and score, only output the best word sequence hypothesis.

-progout

Gradually output the interim results from the first pass at regular intervals.

-proginterval msec

set the -progout output time interval (msec).

-demo The same as "-progout -quiet".

Other

-debug Display debug information.

-C jconffile

Load the jconf settings file. Here runtime options can be loaded that are set in this file.

-version

Display program name, compile time, and compile time options.

-help

Display a brief overview of options.

EXAMPLES

For examples of system usage refer to the Julius documentation.

SEE ALSO

mkbingram(1), adinrec(1), adintool(1), mkdfa(1), mkgsmm(1), wav2mfcc(1)

DIAGNOSTICS

On exiting normally, Julius will return the exit status 0, If an error occurs, Julius exits abnormally, and the exit status 1 is returned.

If an input file cannot be found or cannot be loaded for some reason then Julius will skip processing for that file.

BUGS

There are some restrictions to the type and size of the models Julius can use. For a detailed explanation refer to the Julius documentation.

For bug-reports, inquires and comments please contact Julius@kuis.kyoto-u.ac.jp

AUTHORS

Rev.1.0 (1998/02/20)

Designed by Tatsuya Kawahara and Akinobo Lee

(Kyoto University)

Development by Akinobo Lee (Kyoto University)

Rev.1.1 (1998/04/14)

Rev.1.2 (1998/10/31)

Rev.2.0 (1999/02/20)

Rev.2.1 (1999/04/20)

Rev.2.2 (1999/10/04)

Rev.3.0 (2000/02/14)

Rev.3.1 (2000/05/11)

Development by Akinobo Lee (Kyoto University)

Rev.3.2 (2001/08/15)

Development mainly by Akinobo Lee

(Nara Institute of Science and Technology)

THANKS TO

From Rev.3.2 Julius is released by the "Information Processing Society, Continuous Speech Consortium"

The Windows DLL version was developed and released by Hideki Banno (Nagoya University)

The Windows Microsoft Speech API compatible version was developed by Takashi Sumiyoshi (Kyoto University)

NAME

adinrec - record a one sentence utterance to a file

SYNOPSIS

adinrec [options..] filename

DESCRIPTION

adinrec is a tool that comes with Julius that records one utterance from the microphone device to a file. It observes the input level and if it's greater then a certain level it begins recording. The recorded data is stored in an internal buffer. At the next long pause, recording stops and the buffer is written to a file. If the filename is replaced with a "-" then the output is written to the standard output.

The sampling data type is set to monoral, 16bit signed short (big endian). The output is RAW format (headerless). If a file already exists with the same filename it is overwritten.

OPTIONS

-freq threshold

Sampling frequency in Hz. (default: 16000)

-lv threslevel

Amplitude threshold (0 - 32767). If the amplitude passes this threshold it is considered as the beginning of the speech segment, if it drops below this level then it is treated as the end of the speech segment.
(default: 3000)

-zc zerocrossnum

Zero crossing threshold. (default: 60)

-margin msec

Margin to place at the beginning and end of the speech segment (msec). (default: 300)

-nostrip

Depending on the sound device, invalid "0" samples at the start and end of recording do not have to be removed. The default is automatic removal.

SEE ALSO

julius(1), sox(1), wavplay(1), wavrec(1), aplay(1), arecord(1)

BUGS

For bug-reports inquires and comments please contact julius@kuis.kyoto-u.ac.jp

LICENSE

Under the Julius license.

NAME

mkbingram - make binary N-gram from two arpa LMs

SYNOPSIS

mkbingram 2gramfile rev3gramfile outputfile

DESCRIPTION

mkbingram is a conversion tool to convert a standard ARPA
2-gram and reverse 3-gram files into a single binary
file for use with Julius. Using this binary file decreases
the initialization time of Julius drastically.

mkbingram can load gz compressed ARPA files.

USAGE

When selecting the language model under Julius, instead of
loading the ARPA format files
"-nlr 2gramfile -nrl rev3gramfile", use "-d bingramfile"
to load the binary format file created by mkbingram.

SEE ALSO

julius(1)

BUGS

For bug-reports inquires and comments please contact
julius@kuis.kyoto-u.ac.jp

LICENSE

Under the Julius license.

NAME

mkgshmm - convert monophone HMM to GS (Gaussian Selection) HMM for Julius

SYNOPSIS

mkgshmm monophone_hmmdefs > outputfile

DESCRIPTION

mkgshmm is a perl script to convert a HTK format monophone HMMs into Gaussian Mixture Selection (GMS) HMMs for use with Julius.

From Julius-3.2 the high speed acoustic likelihood calculation technique GMS is supported. Based on the monophone likelihoods for each frame, only a subset of the triphone or PTM states are calculated. This technique speeds up acoustic likelihood calculations by approximately 30%.

EXAMPLE

First prepare a monophone model trained with the same corpus that was used to train the triphone or PTM model to be used.

Next convert these monophones to GMS HMMs using mkgshmm.
(Actually this just involves converting the states to macro definitions)

```
% mkgshmm monophone > gshmmfile
```

Then select this file with Julius option "-gshmm".

```
% julius -C foo.jconf -gshmm gshmmfile
```

Warning: Make sure that the GMS model is created from the same corpus as the triphone or PTM model to be used. If this is not followed then there will be a mismatch with the GMS model, selection errors will occur and performance will degrade.

SEE ALSO

Julius(1)

BUGS

For bug-reports inquires and comments please contact julius@kuis.kyoto-u.ac.jp

LICENSE

Under the Julius License

Bibliography

1. A.Lee, T.Kawahara, and K.Shikano. Julius -- an open source real-time large vocabulary recognition engine. In Proc. EUROSPEECH, pp.1691--1694, 2001.
2. T.Kawahara, A.Lee, T.Kobayashi, K.Takeda, N.Minematsu, S.Sagayama, K.Itou, A.Ito, M.Yamamoto, A.Yamada, T.Utsuro, and K.Shikano. Free software toolkit for Japanese large vocabulary continuous speech recognition. In Proc. ICSLP, Vol.4, pp.476--479, 2000.
3. T.Kawahara, T.Kobayashi, K.Takeda, N.Minematsu, K.Itou, M.Yamamoto, A.Yamada, T.Utsuro, and K.Shikano. Japanese dictation toolkit -- plug-and-play framework for speech recognition R¥&D --. In Proc. IEEE workshop on Automatic Speech Recognition and Understanding, pp.393--396, 1999.
4. T.Kawahara, T.Kobayashi, K.Takeda, N.Minematsu, K.Itou, M.Yamamoto, T.Utsuro, and K.Shikano. Sharable software repository for Japanese large vocabulary continuous speech recognition. In Proc. ICSLP, pp.3257--3260, 1998.

Julius/Julian Release History

'98

02/20	Julius Rev.1.0 release
04/14	Julius Rev.1.1 release
07/30	Julian Rev.1.0 release
10/24	Julius Rev.1.2 release

'99

02/20	Julius Rev.2.0 release
03/01	Julian Rev.2.0 release
04/20	Julius/Julian Rev.2.1 release
10/04	Julius/Julian Rev.2.2 release

2000

02/14	Julius/Julian Rev.3.0 release
05/11	Julius/Julian Rev.3.1 release
06/23	Julius/Julian Rev.3.1p1 release

2001

02/27	Julius/Julian Rev.3.1p2 release
06/22	Julius/Julian Rev 3.2(beta) release
08/18	Julius/Julian Rev 3.2 release

Julius users Mailing List

Julius user’s Mailing List

The Julius user Mailing List is currently only available in Japanese. Please refer to the Japanese documentation for information about this list.

Contact and Links

Links

- Julius/Julian Homepage (Japanese)
- Julius/Julian SAPI version Homepage (Kyoto University) (Japanese)
- Julius for Windows (DLL version) (Hideki Banno Nagoya University)
- Continuous Speech Recognition Consortium (Japanese)

Contacts

- Contact: julius@kuis.kyoto-u.ac.jp
- System Developer: Akinobo Lee (ri@is.aist-nara.ac.jp)

Grammar Based Continuous Speech Parser

Julian

rev. 3.2

(2001/08/18)

Julian

- Julian is a finite state grammar based continuous speech recognition parser. The most likely word sequence depending on the speech input and given grammar is calculated and the result is displayed.
- A two pass A* search is used. In the first pass a beam search using degrees of freedom derived from the grammar takes place. In the second pass the results from the first pass are used as heuristics, and high precision recognition is performed using a A* search.
- Except for the grammar rules, most parts of the system are the same as those used in Julius. The usage, acoustic models that can be used, and speech input settings etc. are the same as Julius (for the same revision).
- Maximum word limit of 65,535 words

New Features in Rev.3.2 and changes from Rev.3.1 to Rev.3.2:

- Word insertion penalty("-penalty1","-penalty2")
- Category constraints considered for interword triphone calculation
- The interword triphone value default has been changed from maximum value to average value (This can be changed with the options "-iwcd1 max","-iwcd1 avg")
- The option "-looktrellis" has been introduced. (default: OFF)

For other new features see Julius-3.2 New Features

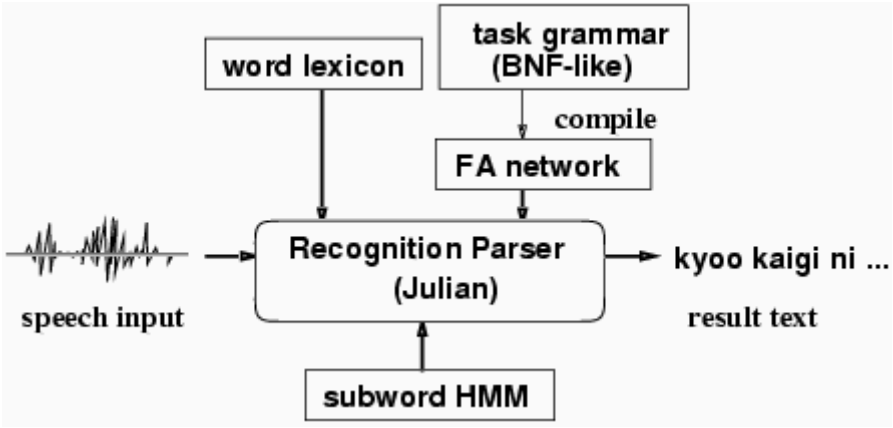
Changes between Rev.2.2 and Rev.3.1:

- Can use PTM models.
- First pass inter-word triphone calculation (approximate, using context dependent triphone likelihoods)

Julian, System Structure

System Structure

The structure of the Julian speech recognition system is shown below



- A finite state grammar language model, and a HMM acoustic model is used.
- The input speech is processed using a two-pass search.
    1. First Pass: Category restricted frame synchronous beam search. (A high speed approximate search)
    2. Second Pass: Grammar based N-best stack decoding. (High Precision)

Operating Environment and Installation

OS

The same operating environments as Julius.

Machine Spec

Minimum Spec : PentiumIII 200MHz, 32MB Usable Memory
Recommended Spec : PentiumIII 600MHz, 128MB Usable Memory

For a several hundred word task, if a monophone model is used, real time processing can occur with a process size under 10MB. For a 5000 word task using PTM then a process size on the order of 30MB is required for real time execution.

Installation

Install Julian using the procedure below.

```
% tar xzvf julius-3.2.tar.gz
% cd julius-3.2
% ./configure --enable-julian
% make
% make install
```

Now Julian will be installed in /usr/local/bin.

NAME

Julian – Grammar Based Continuous Speech Recognition Parser.

SYNOPSIS

julian [-C jconffile] [options ...]

DESCRIPTION

Julian is a continuous speech recognition parser based on finite state grammar. High precision recognition is achieved using a two pass hierarchical search.

Julian can perform recognition on microphone input, audio files, and feature parameter files. Also, as standard format acoustic models and grammar based language models can be used, these models can be changed to perform recognition under various conditions.

The maximum vocabulary is 65,535 words.

Model Usage

Julius uses the following models.

Acoustic Models

Acoustic HMM(Hidden Markov Model) are used. Phoneme models (monophone), context dependent phoneme models (triphone), tied-mixture and phonetic tied-mixture models can be used. When using context dependent models, interword context is taken into consideration. Files written in HTKs HMM definition language are used.

Language Model

For the task grammar, sentence structures are written in a BNF style using word categories as terminating symbols to a grammar file. A voca file contains the pronunciation (phoneme sequence) for all words within each category are created. These files are converted with mkdfa.pl(1) to a deterministic finite automaton file (.dfa) and a dictionary file (.dict)

Speech Input

It is possible to recognize live input from either a microphone A-D or a DatLink (NetAudio) system. Speech waveform files (16bit WAV (no compression), or RAW format) and feature parameter files (HTK format) can be used.

Warning: Julian can only extract MFCC_E_D_N_Z features internally. If it is necessary to use HMMs based on another type of feature extraction then microphone input and speech waveform files cannot be used. Use an external tool such as wav2mfcc to create the appropriate feature parameter files.

Search Algorithm

Recognition in Julian uses a two pass structure. In the first pass a high-speed approximate search is performed using weaker constraints then the given grammar. Here a LR beam search using only inter-category constraints extracted from the grammar is performed.
using the original grammar rules and results from the first pass, the second pass re-searches these, and a high precision result is gained quickly. In the second pass the optimal solution is guaranteed using the A* search.

When using a context dependent phoneme model (triphone), interword contexts considered on both

the first and second passes. For tied-mixture and phonetic tied-mixture models, high speed acoustic likelihood calculations using gaussian pruning are performed.

OPTIONS

The options below allow you to select the models to be used and set system parameters. You can set these option at the command line, however it is recommended that you combine these options in the jconf settings file and use the "-C" option at run time.

Below we give an explanation of each of the options.

Speech Input

-input {rawfile|mfcfile|mic|netaudio|adinserv}

Select the speech wave data input source.
(default: mfcfile)
For information on file formats refer to the Julius documentation.

-NA server:unit

When using (-input netaudio) set the server name and unit ID of the DatLink unit to connect to.

-filelist file

With (-input rawfile|mfcfile) perform recognition on all files contained within the target filelist.

-adport portnum

With (input adinserv) A-D server port number.

Speech segmentation

-pausesegment

-nopausesegment

Force speech segmentation (segment detection) ON / OFF.
(For mic, adinnet default = ON. For files, default = OFF)

-lv threslevel

Amplitude threshold (0 - 32767). If the amplitude passes this threshold it is treated as the beginning of the speech segment, if it drops below this level then it is treated as the end of the speech segment.
(default: 3000)

-headmargin msec

Margin at the start of the speech segment (msec).
(default: 300)

-tailmargin msec

Margin at the end of the speech segment (msec).
(default: 400)

-zc zerocrossnum

Zero crossing threshold. (default: 60)

-nostrip

Depending on the sound device, invalid "0" samples at the start and end of recording may not be removed automatically. The default is to perform automatic removal.

Acoustic Analysis

-smpFreq frequency

Sampling frequency (Hz).
(default: 16kHz = 625ns).

-smpPeriod period

Sampling rate (ns)
(default: 625ns = 16kHz).

-fsize sample

Analysis window size (No. samples).
(default: 400, 25mS)

-fshift sample

Frame shift (No. samples). (default: 160, 10mS)

-hipass frequency

Highpass filter cutoff frequency (Hz).
(default: -1 = disable)

-lopass frequency

Lowpass filter cutoff frequency (Hz).
(default: -1 = disable)

Language Model(BNF type Grammar)

-dfa dfa_filename

Select the finite state automaton grammar file
(.dfa) to use. (Required)

-penalty1 float

First pass word insertion penalty. (default: 0.0)

-penalty2 float

Second pass word insertion penalty.
(default: 0.0)

Recognition Dictionary

-v dictionary_file

Recognition Dictionary File (Required).

-silhead {WORD|WORD[OUTSYM]|#num}

-siltail {WORD|WORD[OUTSYM]|#num}

Sentence start and end silence as defined in the
word dictionary.
(default: "<s>" / "</s>")

These are dealt with specially during recognition to
hypothesize start and end points (margins). They can
be defined as shown below.

	Example
Word_name	<s>
Word_name[output_symbol]	<s>[silB]
#Word_ID	#14

(Word_ID is the word position in the dictionary file starting from 0)

-forcedict

Disregard dictionary errors.
(Skip word definitions with errors)

Acoustic Model(HMM)

-h hmmfilename

The name of the HMM definition file to use.
(Required)

-hlist HMMlistfilename

HMMList filename. Required when using triphone based HMMs. Details are contained in the Julius documentation.
This file provides a mapping between the logical triphones names generated from the phonetic representation in the dictionary and the actual HMM definition names (physical triphones).

-force_ccd / -no_ccd

When using a triphone acoustic model these options control interword context dependency. If neither of these options are set then the use of interword context dependency will be determined from the models definition names.
If the "-force_ccd" option is set when using something other then a triphone model, there is no guarantee that Julius will run.

-notypecheck

Do not check the input parameter type.

(default: Perform the check)

-iwcd1 {max|avg}

When using a triphone acoustic model set the
interword acoustic likelihood calculation method
used in the first pass.

max: The maximum same context triphone value (default)

avg: The average same context triphone value

Options for tied-mixture and PTM acoustic models

-tmix K

Perform Gaussian Pruning only calculate the upper
k gaussian densities per codebook. (default: 2)

-gprune {safe|heuristic|beam|none}

Set the gaussian pruning technique to use.
(default: safe (standard) beam (high-speed))

-gshmm hmmdefs

Set the Gaussian Mixture Selection monophone
model to use. A GMS monophone model is generated
from an ordinary monophone HMM model using the
attached program mkgshmm(1).
(default : none (do not use GMS))

-gsnum N

When using GMS, only perform triphone calculations
for the top N monophone states. (default: 24)

Search Parameters (First Pass)

-b beam_width

Beam width (Number of HMM nodes).
As this value increases, precision also increases,
however, processing time and memory usage also
increase.

default values: Model dependent,
400 (monophone)

800 (triphone,PTM)
1000 (triphone,PTM,engine=v2.1)

-1pass

Only perform the first pass search. This mode is automatically set when no 3-gram language model has been specified (-nlr).

-realtime

-norealtime

Explicitly state whether real time processing will be used in the first pass or not. For file input the default is OFF (-norealtime), for microphone, or NetAudio network input the default is ON (-realtime). This option relates to the way CMN is performed: when OFF CMN is calculated for each input independently, when the realtime option is ON the previous 5 second of input is always used. Refer to -progout.

Search Parameters (Second Pass)

-b2 hyponum

Hypothesis envelope width. This number of hypotheses are expanded (sorted by length), shorter hypothesis are not expanded. This prevents search failures. (default: 30)

-n candidate_num

The search continues until "candidate_num" sentence hypothesis have been found. These hypotheses are re-sorted by score and the final result is displayed. (Refer to the "-output" option). As Julius does not strictly guarantee a optimal second pass search, the maximum likelihood candidate is not always given first.

As this value is increased the probability that the maximum likelihood hypothesis is returned increases,

but as a prolonged search must be performed, the
processing time also becomes large. (default: 1)

default value is dependent on the recognition engine
settings ("--enable-setup= ").
10 (standard)
1 (fast,v2.1)

-output N

Used with the "-n" option above. Output the top N
sentence hypothesis. (default: 1)

-sb score

Score envelope width. For each frame, do not scan
parts that deviate from the highest score by more
then this envelope. This directly relates to the speed
of the second pass acoustic likelihood calculations.
(default: 80.0)

-s stack_size

The maximum number of hypothesis that can be stored
on the stack during the search. A larger value gives more
stable results, but increases the amount of memory
required. (default: 500)

-m overflow_pop_times

Number of expanded hypotheses required to
discontinue the search. If the number of expanded
hypotheses is greater then this threshold then, the search
is discontinued at that point. The larger this
value is, the longer the search will continue, but
processing time for search failures will also
increase. (default: 2000)

-lookuprange nframe

When performing word expansion, this option sets
the number of frames before and after in which to consider
word expansion. This prevents the omission of short

words but, with a large value, the expansion hypotheses increase and becomes slow. (default: 5)

Forced alignment

-walign

Return the result of viterbi alignment of the word units from the recognition results.

-palign

Return the result of viterbi alignment of the phoneme units from the recognition results.

Message Output

-quiet

Omit phoneme sequence and score, only output the best word sequence hypothesis.

-progout

Gradually output the interim results from the first pass at regular intervals.

-proginterval msec

set the -progout output time interval (msec).

-demo The same as "-progout -quiet".

Other

-debug Display debug information.

-C jconffile

Load the jconf settings file. Here runtime options can be loaded that are set in this file.

-version

Display program name, compile time, and compile time options.

-help

Display a brief overview of options.

EXAMPLES

For examples of system usage refer to the Julian documentation.

SEE ALSO

mkbingram(1), adinrec(1), adintool(1), mkdfa(1),
mkgsmm(1), wav2mfcc(1)

DIAGNOSTICS

On exiting normally, Julian will return the exit status
0, If an error occurs, Julian exits abnormally, and the
exit status 1 is returned.

If an input file cannot be found or cannot be loaded for
some reason then Julian will skip processing for that file.

BUGS

There are a number of restrictions to the type and size of the
models Julian can use. For a detailed explanation refer
to the Julian and Julius documentation.

For bug-reports, inquires and comments please contact
julius@kuis.kyoto-u.ac.jp

AUTHORS

Rev.1.0 (1998/07/20)

Designed by Tatsuya Kawahara and Akinobo Lee

(Kyoto University)

Rev.2.0 (1999/02/20)

Rev.2.1 (1999/04/20)

Rev.2.2 (1999/10/04)

Rev.3.1 (2000/05/11)

Development by Akinobo Lee (Kyoto University)

Rev.3.2 (2001/08/15)

Development mainly by Akinobo Lee

(Nara Institute of Science and Technology)

THANKS TO

Up to Rev.3.1 this program was released under the speech media laboratory, Kyoto University (Doshiya Lab). From Rev.3.2 Julian has been integrated with Julius and released under the "Information Processing Society, Continuous Speech Recognition Consortium".

The Windows Microsoft Speech API compatible version was developed by Takashi Sumiyoshi (Kyoto University).

Writing Grammar For Julius

Here an outline is given on how to write grammar to be used with Julian. For detailed information refer to the separately distributed "Julian Grammar Kit" toolkit.

Task Grammar

Julian differs from Julius in that the sentence structure and words to be recognized have been explicitly written. Julian performs a maximum likelihood search using only the degrees of freedom given by the grammar, and outputs the word sequence that best matches the input utterance.

In most systems the sentence structure and word list are written for the specific task. Here we call this "task grammar".

Describing Task Grammar

The task grammar is described in two files. A BNF style grammar file describes sentence structure and category rules, and a voca file which registers word declarations and pronunciations (phoneme sequence) for each category.

Using the sentence structure constraints from the format above it should be possible to describe CFG classes however, Julian can only handle regular grammar classes. This constraint is automatically checked at compile time. Also only left recursion can be handled. As an example, think of the grammar that will describe the sentences below.

- "Make it white"
- "Change it to red"
- "Make it red"
- "Quit"

The grammar file below can be written to describe these sentences. The symbol S is fixed as the sentence start symbol. NS_B, NS_E, and NOISE correspond to silences (pauses) at the start, end and within a sentence.

```
S           :       NS_B CHGCOLOR_S NS_E
S           :       NS_B QUIT_S NS_E
CHGCOLOR_S  :       MAKE_V OBJECT NOISE COLOR_NP
CHGCOLOR_S  :       CHANGE_V OBJECT TO NOISE COLOR_NP
COLOR_NP    :       COLOR_N
QUIT_S      :       QUIT_V
```

Symbols that haven't appeared on the left are terminal symbols (shown above in red), in other words they are word categories. Words pertaining to each of these word categories are registered in the voca file.

```
% COLOR_N
RED             r E d
WHITE                   w Y t
GREEN                   g r i n
BLUE            b l u
% OBJECT
IT              I t
% TO
TO              t u
% MAKE_V
MAKE                    m e k
% CHANGE_V
CHANGE                  C e n J
% QUIT_V
QUIT            k w I t
% NS_B
silB            silB
% NS_E
silE            silE
% NOISE
sp              sp
```

Conversion to Julian Format

Convert the grammar and voca files into a deterministic finite automaton file (dfa) and dictionary file (dict) using the compiler "mkdfa.pl".

```
(When the grammar file is sample.grammar, and the voca file is sample.voca)
% mkdfa.pl sample
sample.grammar has 6 rules
sample.voca       has 9 categories and 12 words
---
Now parsing grammar file
Now modifying grammar to minimize states[1]
Now parsing vocabulary file
Now making nondeterministic finite automaton[10/10]
Now making deterministic finite automaton[10/10]
Now making triplet list[10/10]
---
-rw-r--r--    1 foo        users          134 Aug 17 17:50 sample.dfa
-rw-r--r--    1 foo        users          212 Aug 17 17:50 sample.dict
-rw-r--r--    1 foo        users           75 Aug 17 17:50 sample.term
```

Starting Julian

Using these converted files, start Julian as shown below.

```
% julian -dfa sample.dfa -v sample.dict...
```

Apart from the grammar it is also necessary to select the acoustic model to be used. Search parameters can also be set. These settings can be stored in a jconf setting file.

These options are set the same way as for Julius.