The Julius book

Akinobu LEE

May 17, 2010

The Julius book
by Akinobu LEE

Edition 1.0.3 - rev.4.1.5

Contents

List of Tables

まえがき

Julius は，音声認識システムの開発・研究のためのオープンソースの高性能な汎用大語彙連続音声認識エンジンである．数万語彙の連続音声認識を一般の PC 上でほぼ実時間で実行できる．また，高い汎用性を持ち，発音辞書や言語モデル・音響モデルなどの音声認識の各モジュールを組み替えることで，様々な幅広い用途に応用できる．

エンジンの中心部は，組み込み可能なライブラリの形で提供されており，一般アプリケーションに音声認識機能を組み込めるよう設計されている．また，プラグインを使って機能拡張することも可能である．Julius は言語非依存であり，対象言語の音響・言語モデルさえあればその言語の認識器として動作することができる．

Julius はオープンソースソフトウェアであり，ソースコードを含めて誰でも無償で入手することができる．ライセンスは商用も可能としている．

Julius の開発目的は，近年の音声認識技術進展の成果を一般に広く公開すること，および，共通のプラットフォームを提供することによって音声関連の研究やアプリケーション開発を広く促すことである．最初のバージョンは 1996 年に公開され，それ以来，音声認識技術の進歩へのキャッチアップや機能追加，リファインなどのために現在も開発が継続的に行われている．

Julius の研究・開発に関わっている主な機関は以下のとおりである．コンタクトは，Julius のオフィシャルの管理者 ML julius-info at lists.sourceforge.jp へメールを送るか，あるいは各機関や作者に直接問い合わせてもよい．

Web ページは http://julius.sourceforge.jp にある．Julius の配布のほか，解説文書やチュートリアル，ソースコードのリファレンスデータベース，および開発者が情報交換できる Web フォーラムがある．開発途中の最新版のスナップショットも CVS で入手可能である．Web フォーラムではバグ情報や更新情報，QA などの全般的な情報がやりとりされており，一読をお勧めする．

Julius の名前の由来については，"Japanese Utterance Listening, Indexing and Understanding System" の頭文字をとったという説と，"Dictation"（口述筆記）→ "Dictate"（動詞：支配する）→ "Julius Caesar"（ジュリアス・シーザー）→ "Julius" という連想から名付けられたという説がある．表記については，初期のころは文献ごとにゆれが見られ，"JULIUS" とすべて大文字で表記される場合もあったが，現在は "Julius" が正式な表記であるとされている．日本語での発音は「ジュリアス」「ジュリウス」「ユリアス」「ユリウス」など諸説あるが，開発チーム内では「ジュリアス」が用いられている．バージョン 3.x までは文法を用いるバージョンが "Julian" と別名で呼ばれていたが，バージョン 4 で Julius に統合・吸収された．

この文書は，音声認識エンジン Julius のトータルリファレンスである．インストールから音声入力の仕様，各モデルの具体的仕様，オプション一覧，ライブラリ・プラグインの解説などが書かれている．本書が読者の助けになれば幸いである．

Chapter 1

概要

本章では，Julius の動作環境やパッケージの概要について述べる．

1.1 動作環境

Julius は C 言語で書かれており，様々なプラットフォームで動作する．基本となる開発環境は Linux および Windows である．そのほか，Solaris, FreeBSD および MacOSX でも動作する．その他の OS では，ファイル入力では動作するが，デバイスからの直接入力がサポートされていない（ただし音声入力プラグインによって拡張可能）．原理的には，Windows Mobile や iPhone などのスマートフォンや，組み込みマイコンでの動作も可能である．

音声デバイスからの入力を直接認識するには，録音環境についていくつかの要件がある．音声データ入力の章を参照のこと．

1.2 音声認識システムの実行に必要なもの

Julius は単体では動作しない．Julius を動かすには，

- 認識対象とする発声文の単語単位のパターンを決める「言語モデル」
- 音素ごとの音声波形パターンを統計的にモデル化した「音韻モデル」

の 2 つが必要である．Julius は言語モデルとして，統計言語モデルである「単語 N-gram」，人手で規則を記述する「認識用文法」，および辞書のみを用いた 1 単語発声の単語認識をサポートする．また音響モデルとしては，HTK で学習されたサブワード単位の Hidden Markov Model (HMM) をサポートする．

モデルの仕様やサポート範囲，使いかたの詳細については，それぞれの章を参照のこと．また，モデルの入手性については Julius の Web ページを参照のこと．Web ページでは日本語の標準モデルを配布している．[1]

1.3 パッケージの構成

ソースパッケージとバイナリパッケージが配布されている．どちらも tar.gz 形式でホームページからダウンロードできる．バイナリパッケージは，コンパイル済みの実行バイナリと関連ファイルのみのパッケージであり，Linux と Windows コンソール用が用意されている．その他，ホームページからは最新の開発版スナップショットを CVS で取得することも可能である．

Julius の配布パッケージには，以下のツールが含まれている．

- julius --- 連続音声認識エンジン Julius 本体
- adinrec --- 録音テストツール
- adintool --- 汎用音声入出力サーバ

[1] なお，英語のモデルは存在するが，学習元である音声データベースのライセンス上の制約があり，精度にも改善の余地があることから，無償公開はしていない．

- jcontrol --- モジュールモード用サンプルクライアント (C)
- jclient.pl --- モジュールモード用サンプルクライアント (Perl)
- mkbingram --- バイナリ N-gram 作成コマンド
- mkbinhmm --- バイナリ HMM 作成コマンド
- mkbinhmmlist --- バイナリ HMMList 作成コマンド
- mkgshmm --- GMS 用音響モデル作成ツール
- mkss --- スペクトルサブトラクション用ノイズスペクトル推定ツール
- 言語モデル・認識文法関連ツール --- mkdfa.pl, mkfa, dfa_determinize, dfa_minimize, accept_check, nextword, generate, generate-ngram, gram2sapixml.pl, yomi2voca.pl

また，Linux ではインストール時には以下のライブラリとヘッダ，および関連ツールがインストールされる．

- `libsent.a` --- 汎用ライブラリ
- `libjulius.a` --- 認識エンジンコアライブラリ
- `include/sent/*` --- libsent 用ヘッダ
- `include/julius/*` --- libjulius 用ヘッダ
- `libsent-config`, `libjulius-config` --- ライブラリ組み込み時に必要なコンパイラ設定を返すスクリプト

1.4 処理フロー

Julius の処理フローの概要を以下に示す．音声入力部・特徴抽出部・認識処理部（第 1 パス）は，ファイル入力の場合は 1 入力ごとに各部が順番に実行されるが，マイク入力などでリアルタイム認識を行う場合，各部が並行処理される．

認識処理全体は 2 パス構成となっており，第 1 パスでは，入力と並行して認識処理が行われる．第 2 パスでは，第 1 パスの結果である「単語トレリス」と呼ばれる仮説集合を参照しながら，入力全体に対して再認識を行い結果を確定する．各部の詳細はそれぞれの対応する章を参考のこと．

Chapter 2

インストール

本章では Julius のコンパイルおよびインストールについて仕様を述べる．まずバイナリパッケージについて説明した後、Linux や互換環境 (cygwin / mingw) における一般的なコンパイル方法とオプション設定方法について述べる。その後、OS や環境ごとのコンパイル方法について詳しく述べる。

2.1 バイナリパッケージ

Julius はコンパイルずみのバイナリパッケージとしても公開されている．パッケージを展開すると，bin ディレクトリ以下に実行バイナリが含まれており，直接実行できる．Julius は実行バイナリのみで動作するため，システムへのインストールは必須ではない．

バイナリパッケージからシステムへインストールする場合は，`bin`, `include`, `lib`, `doc` の各ディレクトリの内容を手動でコピーする．例として `/usr/local` にインストールする場合の各ディレクトリのコピー先を以下に示す．

```
bin/* -> /usr/local/bin
include/* -> /usr/local/include
lib/* -> /usr/local/lib
doc/man/man1/* -> /usr/local/man/man1
doc/man/man5/* -> /usr/local/man/man5
```

2.2 ソースからコンパイルする

Julius は configure スクリプトに対応している．Linux / cygwin / mingw 環境では、ソースコードのアーカイブをダウンロードして展開した後，展開したディレクトリにおいて

```
% ./configure
% make
```

を実行することでコンパイルが行える．また，

```
% make install
```

でインストールできる．

使用するコンパイラは環境変数 `CC` で，コンパイラのオプションは環境変数 `CFLAGS` で変更できる．コンパイラとして **cc** を用い，オプションとして `-O3` を用いる場合の例を以下に示す．

```
% export CC=cc
% export CFLAGS=-O3
% ./configure
```

make install を行うと，Julius および関連ツールの実行バイナリ，ライブラリおよびヘッダファイルが以下のようにコピーされる．なお，インストール先のトップディレクトリを`${prefix}`とする．

```
bin -> ${prefix}/bin
include -> ${prefix}/include
lib -> ${prefix}/lib
doc/man/man1 -> ${prefix}/man/man1
doc/man/man5 -> ${prefix}man/man5
```

${prefix} は通常 /usr/local である．これは **configure** オプション--prefix=... で変更できる．たとえば，ユーザのホームで使用する場合は以下のように実行する．

```
% ./configure --prefix=/home/foobar/julius/build
```

configure スクリプトは GNU autoconf を用いて作成されたものであり，上記の --prefix を含めた標準的な configure オプションが使用できる．また，設定や認識アルゴリズムを変更できる Julius 独自の configure オプションも用意されている．

2.3 コンパイル時オプション

configure に指定可能な Julius 独自のオプション一覧を以下に示す．libsent オプション，libjulius オプション，および julius オプションの 3 つのグループに分類されているが，それぞれ，サブディレクトリ libsent（汎用ライブラリ）, libjulius（認識エンジン本体）および julius（アプリケーションとしての Julius）の下にある configure に渡される．トップディレクトリの configure に渡されたオプションはサブディレクトリの全ての configure へ引き渡されるので，コンパイル時に指定する際は，区別せずにトップディレクトリの configure に与えばよい．

2.3.1 libsent オプション

libsent は Julius のための汎用ライブラリであり，入出力や特徴量抽出，音響モデル，言語モデル，出力確率計算などの関数が定義されている．

--enable-words-int 単語 ID の内部形式を unsigned short から int に変更し，語彙サイズが 65,535 以上の言語モデルを扱えるようにする．ただし使用メモリ量は増大する．

--enable-msd 音響モデルにおいて MSD-HMM のサポートを有効にする．

--disable-class-ngram クラス N-gram 対応を無効にする．メモリを数十 KByte 節約できる．

--enable-fork ネットワーク入力（-input adinnet）指定時に，クライアントから接続を受けたときに自らを fork するようにする．デフォルトは fork しない．

--with-mictype={auto|oss|alsa|esd|portaudio|sp|freebsd|sol2|sun4|irix} マイク入力で使用する API を指定する．指定なし，あるいは auto を指定した場合は自動判別される．自動判別の場合のデフォルトは，Linux においては alsa, oss, esd の順で最初に見つかったもの，Windows では portaudio となる．Windows の portaudio では，コンパイル時に DirectSound のヘッダがある場合は DirectSound, ない場合は WMM ドライバが選択される．4.1.3 以降では、システムに portaudio ライブラリとヘッダがインストールされている場合はそちらを使い、なければ Julius 内蔵の portaudio (V18) を使う。

--with-netaudio-dir=dir DatLink からの音声入力を使用する場合に，DatLink 付属の NetAudio ライブラリの include と lib があるディレクトリを指定する．

--disable-zlib 圧縮ファイルの読み込みにおける zlib の使用を無効化する．デフォルトでは，zlib があれば使用し，なければ gzip コマンドを内部で呼び出す．

--without-sndfile libsndfile を使用しない．

2.3.2 libjulius オプション

libjulius は認識エンジンのコアライブラリであり，実際の認識処理を担当する．--enable-setup の項目以降はアルゴリズムの選択に関するオプションであり，通常は設定する必要はあまりない．

--enable-gmm-vad GMM は通常，入力を音声認識した後，その入力全体について音声・非音声を判断し，非音声であれば棄却するという入力棄却に用いられる．このオプションを指定することで，GMM を入力棄却ではなく，認識の前段階でフレーム単位で音声区間を検出する VAD に用いることができる．詳細は音声区間検出の章を参照のこと．

--enable-decoder-vad デコーダの認識結果に基づく VAD を有効にする．`-spsegment`（ショートポーズセグメンテーション）指定時に有効になる．詳細は，音声区間検出の章および探索アルゴリズムの章を参照のこと．

--enable-power-reject 入力全体の平均パワーに基づく入力棄却を有効にする．パワー項を持つ特徴量を使用する場合のみ使用できる．音声区間検出・入力棄却の章を参照のこと．

--enable-setup={standard|fast|v2.1} 認識アルゴリズムの設定を以下の 3 種類のプリセットから選択できる．

- `fast`: 高速設定（デフォルト）
- `standard`: 標準設定
- `v2.1`: 標準設定

`fast`（高速設定）は，速度を中心に調整した設定である．アルゴリズムは近似計算や探索の枝刈りが多く導入された高速寄りの設定となる．具体的には，Gaussian pruning 法のデフォルトは最も足切り性能の高い beam pruning となり，木構造化辞書の線形化は N-gram の上位頻度の単語について行われる．また，第 2 パスの探索中の単語接続部のトライフォン計算は効率のために遅延計算される．

`standard` は，精度を重視した設定である．アルゴリズムは近似誤差や探索の枝刈り誤りを抑えた設定となるが，処理時間はそれだけかかるようになる．具体的には，Gaussian Pruning は枝刈りエラーを生じない safe pruning がデフォルトとなる．木構造化辞書の線形化は，二音素以下の長さの短い単語すべてについて行われる．また，第 2 パスの探索中に遅延なしに厳密な単語間 triphone を計算する．

`v2.1` は古い Julius の Rev.2.1 と同等のアルゴリズムに戻す．全てのアルゴリズムオプションが無効化される．このオプションは通常は用いず，後述の詳細な設定を用いるためにすべてのオプションをいったん無効化 OFF にしたい場合に使用する．

各設定でセットされるオプションを表にまとめると以下のようになる．表中の ○ のオプションが実際に configure 内で enable される．

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

--enable-factor2 第 1 パスの探索において 2-gram factoring を使う．デフォルトは 1-gram factoring を使う．第 1 パスの認識精度が若干改善するが，最終的な認識結果への影響は小さいとされる．またメモリ量および計算時間が多くかかるようになる．

--enable-wpair 第 1 パスの探索における単語履歴近似について，デフォルトの 1-best 近似に代わって単語対近似を用いる．第 1 パスの精度が向上するが最終的な認識結果への影響は比較的小さいとされる．直前単語ごとに辞書のコピーを内部で用意するため，メモリ使用量は大幅に増大する．また処理速度も遅くなる．

--enable-wpair-nlimit `--enable-wpair` と同時に指定することで，依存して木構造化辞書をコピーする直前単語の最大数を指定できる．

--enable-word-graph 第 1 パスから第 2 パスへ引き渡す中間結果として，単語グラフ形式を用いる．デフォルトは単語トレリス形式である．これは実験用のオプションである．

--disable-pthread pthread による音声取り込み部のスレッド化サポートを無効化する．

--disable-plugin プラグイン機能を無効化する．

2.3.3 julius オプション

アプリケーションとしての Julius は，上記のライブラリを組み込んで動作する，スタンドアロンの認識アプリケーションである．モジュールモード（クライアントサーバ），入力の保存，文字コード変換などアプリケーションよりの機能はこちらに実装されている．

`--enable-charconv={auto|iconv|win|libjcode|no}` 出力テキストの文字セットを変換するライブラリを指定する．`iconv` は iconv ライブラリを，`libjcode` は Julius 内部の libjcode ライブラリを使用する（日本語のみサポート）する．`win` は Windows のネイティブ API を使用する．`no` は文字コード変換そのものを無効にする．デフォルトは `auto` であり，Linux では iconv があればそれを使用し，無ければ libjcode となる．Windows 環境ではデフォルトで win が用いられる．

2.4 コンパイル手順の詳細

OS ごとのコンパイル方法について以下に示す.

2.4.1 Linux

Julius をコンパイルするには，通常の C 言語の開発環境のほかに以下の開発環境（ライブラリとヘッダ）が必要である．

- `zlib`
- `flex`

また，以下のものについてもあらかじめインストールしておくことを推奨する（無くてもコンパイルは可能）．

- ALSA ヘッダとライブラリ（ALSA での音声入力サポート，無い場合は OSS 使用）
- ESD ヘッダとライブラリ（ESD 経由の音声入力サポート）
- `libsndfile`（音声ファイル形式を拡張）

ほとんどの Linux ディストリビューションで上記はパッケージとして提供されている．例えば Debian GNU/Linux 4.0 Etch では，以下のコマンドをルート権限で実行することで必要な環境を構築できる．

```
# aptitude install build-essential zlib1g-dev flex
# aptitude install libasound2-dev libesd0-dev libsndfile1-dev
```

2.4.2 Windows - cygwin

cygwin 上で Julius をコンパイルするには、以下のパッケージが必要である。

```
Devel
    - binutils
    - flex
    - gcc-core
    - gcc-mingw-core
    - libiconv
    - make
    - zlib-devel
Utils
    - diffutils
Perl
    - perl
MinGW
    - mingw-zlib
```

サウンド取り込み API として DirectSound を使用するため、音声入力を使用するにはコンパイルの際に DirectSound API のヘッダが必要である。DirectSound 用のヘッダは Microsoft DirectX SDK（無償）に含まれているので、あらかじめ入手し、その中にある以下のファイル cygwin の `/usr/include` と `/usr/include/mingw` の両方にそれぞれコピーしておく。

```
d3dtypes.h
ddraw.h
dinput.h
directx.h
dsound.h
```

コンパイル時にこれらのファイルが見つからない場合、Julius は古いサウンド API である MMLIB を使用するよう設定されるが、入力の遅延がかなり大きい等の問題があり、お勧めしない。以下のように julius を実行することで、DirectSound で正しくコンパイルされているかどうかを調べられる。

```
% julius.exe --version
...
primary A/D-in driver: pa-dsound (PortAudio ....)
...
```

コンパイルは通常の Linux 版と同様にコンパイルする。cygwin 環境の無いマシンでも動作する実行バイナリを作る場合は、コンパイラに `-mno-cygwin` をつける。これは以下のように **configure** を実行すればよい。

```
% CC="gcc -mno-cygwin" configure
```

2.4.3 Windows - mingw

Julius をフリーの開発環境である MinGW (Minimalist GNU for Windows) 上でコンパイルすることができる．MinGW, MSYS, および msys-DTK がインストールされている必要がある．

Win32 用の zlib のライブラリとヘッダ，flex ライブラリ，および音声入力のために DirectSound 用のヘッダが必要である．これらは MinGW のディストリビューションには含まれていないので，それぞれ別途入手し，`/mingw/lib/` および `/mingw/include/` 以下に置いておく必要がある．以下に必要なファイルの一覧を示す．

```
/mingw/include/d3dtypes.h
/mingw/include/ddraw.h
/mingw/include/dinput.h
/mingw/include/directx.h
/mingw/include/dsound.h
/mingw/include/zconf.h
/mingw/include/zlib.h
/mingw/lib/libfl.a
/mingw/lib/libz.a
```

コンパイルおよびインストールは，Linux と同様に以下のコマンドで行える．

```
% ./configure
% make
% make install
```

2.4.4 Windows - Microsoft Visual C++

4.1.3 以降の Julius は Microsoft Visual C++ (以下 MSVC) でのコンパイルをサポートしている。MSVC および C++の開発環境のために、JuliusLib のラッパークラスの定義、および Windows サンプルアプリケーション SampleApp が 4.1.3 より同梱されている。ここではそれらのコンパイル方法と使用方法について概説する。

サポートする MSVC のバージョンは 2008 のみである。Professional Edition および Express Edition でのコンパイルを確認している。Windows XP, Vista, 7 で動作確認している。

2.4.4.1 準備

Microsoft DirectX SDK が必要である。Microsoft のウェブサイトから入手できるのであらかじめダウンロード・インストールしておく。

MSVC でコンパイルする場合、Julius は以下の 2 つのオープンソースライブラリを使用する

- zlib
- portaudio V19

いずれもコンパイル済みのファイルが msvc/zlib と msvc/portaudio に含まれている。動作しない場合はそれぞれのライブラリを再構築して、各フォルダ内のファイルを入れ替える。また、portaudio を再構築した場合はその dll を Julius の実行するフォルダ（"Release" と "Debug"）のものに上書きすること。

2.4.4.2 コンパイル

msvc フォルダ内にあるソリューションファイル JuliusLib.sln を開き、ビルドを以下の順で実行する。

- libsent
- libjulius
- julius
- SampleApp

これでコンソール版 Julius ("julius.exe") および GUI 版サンプルアプリ SampleApp ("SampleApp.exe") の実行バイナリが生成される。

2.4.4.3 テスト実行

julius.exe は Win32 のコンソールアプリであり、Linux や cygwin 版と同様にコンソール上で動作する。

SampleApp.exe は GUI 版のサンプルである。JuliusLib およびその VC++向けラッパークラスを使用する。テストするには、アプリケーションを起動後、メニューから jconf ファイルを開き、次に同じくメニューからエンジンの実行を指定する。Julius エンジンはメインウィンドウの子スレッドとして動作し、音声入力開始や認識結果出力などの各イベントをメインメッセージに描画する。Julius エンジンのエラーは同じフォルダの juliuslog.txt に出力される。

2.4.4.4 MSVC でのコンパイル時オプションの設定方法

MSVC でのコンパイルでは **configure** スクリプトを使用しない。その代わりに、**configure** で自動的に設定されるはずの各種 define が、msvc/config フォルダ内の以下のファイルに収められている。

- config-msvc-julius.h
- config-msvc-libjulius.h
- config-msvc-libsent.h

前節で述べたコンパイル時オプションを MSVC で設定する場合、各設定に対応する define を上記のファイル内で直接指定することになる。対応については、まず linux か cygwin で所望のオプションをつけて **configure** を実行し、生成される以下の各ファイルと内容が同じになるようにすればよい。

- julius/config.h
- libjulius/config.h
- libsent/libsent.h

Chapter 3

音声データ入力

Julius は録音済みの音声波形データに対して認識処理が行える．また，音声入力デバイスから音声を直接取り込みながらオンライン認識を行うこともできる．あらかじめ抽出した特徴ベクトル列ファイルを入力とすることもできる．これらの音声入力の選択は，`-input` オプションで行う．本章では音声データの入力について述べる．

なお，説明中に出てくる Julius の各オプションの詳細は，リファレンスマニュアルの「オーディオ入力」の項目も参考にすることを推奨する．

3.1 基本フォーマット

音声波形入力は，あらかじめ録音された音声波形ファイルを入力として与える方法のほか，録音デバイスからの直接音声入力やネットワーク経由の音声受信を行うことができる．以下に与えることのできる音声波形データに共通の仕様について述べた後，各入力方法について詳しく述べる．

3.1.1 ビット数

量子化ビット数は，16 ビット固定である．

3.1.2 チャンネル数

チャンネル数は，1 チャンネル（モノラル）のみサポートする．

3.1.3 サンプリングレート

入力のサンプリングレート (Hz) は，オプション `-smpFreq` あるいは `-smpPeriod` で指定できる．また，`-htkconf` で HTK Config ファイルを与えた場合，その中の `SOURCERATE` の値からセットされる．無指定時のサンプリングレートのデフォルトは 16,000 Hz である．

使用する音響モデルの学習条件に合わせてサンプリングレートを設定する必要がある．入力のサンプリングレートが音響モデルの学習データのレートと一致しない場合，うまく認識できない．たとえば，使用する音響モデルが 16kHz のデータで学習されたものである場合，Julius が取り込む音声入力も 16kHz である必要がある．

また，複数の音響モデルを用いる場合，すべての音響モデルに対して同一のサンプリングレートをそれぞれ指定する必要がある．これは，複数の音響モデルは一つの音声入力を共有するためである．この場合，サンプリングレートはそれぞれの音響モデルごとに (オプション `-AM` のあとに) 同じ値を指定する必要がある点に注意されたい．また GMM については `-AM_GMM` で指定する．なお，複数モデル認識では，サンプリングレートの他に，分析条件の `-fshift` および `-fsize` も同一である必要がある．詳しくは次章の特徴量抽出の節を参照のこと．

マイクロフォンなどのキャプチャデバイスから直接音声を取り込む場合，上記で指定されたサンプリングレートが録音デバイスにセットされ，そのサンプリングレートでの録音が開始される．一方，ファイル入力の場合は，そのファイルのサンプリングレートが指定値と一致するかチェックされ，一致しない場合はエラーとなる．[1]

[1] ただし，RAW 形式のファイル入力やネットワーク入力の場合，入力音声データのヘッダ情報が無いためこのチェックが行われない．

サンプリングレート変換については，48kHz から 16kHz へのダウンサンプリングがサポートされている．オプション `-48` を指定することで，48kHz で入力を取り込み，Julius 内部で 16kHz へダウンサンプリングしながら認識を行える．

3.2 ファイル入力

`-input rawfile` を指定すると，音声波形ファイルの認識が行える．起動後にプロンプトが出るので，音声波形ファイルのパス名を与えると，そのファイル内の音声データに対して認識を行う．認識終了後は，プロンプトに戻る．

デフォルトでは，1 ファイルを 1 発話として認識を行う．このため，入力ファイルはあらかじめ発話単位で区切られている必要がある．1 ファイルの最大長は 320,000 サンプル（16kHz サンプリングで 20 秒）である．[2] また，Julius は最初と最後に一定長の無音が存在するという前提で認識処理を行うので，ファイルの先頭と末尾には数百ミリ秒程度の無音区間があることが望ましい．

ファイル中の無音区間を検出し，そこを区切れ目として連続的に認識を行うことが可能である．これにはオプション `-cutsilence` を指定する．切り出しのアルゴリズムはマイク入力の場合と同一であり，振幅と零交差数により音声区間を切り出す．また，GMM に基づく VAD を用いて音声区間のみを認識することもできる．

単語 N-gram を用いた認識では，ショートポーズセグメンテーション (`-spsegment`) を指定すれば，講演音声のような無音をほとんど挟まない連続発話であっても，短いポーズで細かく区切りながら逐次認識することができる．

複数のファイルをバッチ的に連続認識させるには，認識したいファイル名を一行に 1 つずつ記述したテキストファイルを用意し，オプション`-filelist` で与える．

3.2.1 サポートするファイル形式

読み込み可能なファイル形式は，デフォルトでは以下のとおり．

- .wav ファイル：Microsoft WAVE 形式 WAV ファイル（16bit, 無圧縮 PCM, monoral のみ）
- ヘッダ無し RAW ファイル：データ形式は signed short (16bit)，Big Endian, monoral

また，コンパイル時に Julius に libsndfile を組み込むことで，上記に加えて AU, AND, NIST, ADPCM などの他の形式のファイルを読みこめる．libsndfile は Julius をソースからコンパイルする際に libsndfile のライブラリおよびヘッダがあれば自動的に組み込まれる．

RAW ファイル形式はいくつか注意が必要である．Julius では RAW 形式に対して Big Endian バイトオーダを前提としており，Little Endian のデータは正しく読み込むことができない．16bit,mono の RAW ファイルの Little Endian から Big Endian への変換は，たとえば sox を用いて以下のように行うことができる．

```
% sox -t .raw -s -w -c 1 infile -t .raw -s -w -c 1 -x outfile
```

なお，RAW ファイルには対してはデータ形式やサンプリングレートのチェックが行われないため，入力データが上記の形式に沿っているか，所望のサンプリングレートで録音されたものか，の点に注意すること．

3.3 録音デバイスからの直接入力

オプション `-input mic` を指定することで，マイクロフォンやライン入力などの録音デバイスから音声ストリームをキャプチャしながら認識できる．この直接入力は，Linux, Windows, Mac OS X, FreeBSD および Solaris でサポートされている．

直接入力では，入力ストリームに対して音声区間の切り出しが行われ（デフォルトではレベルと零交叉に基づく検出が行われる），検出された音声区間ごとに認識処理を行う．また，特徴量抽出および第 1 パスの認識処理は入力と並行してリアルタイムに行われる．音声区間検出の調整やリアルタイム認識の注意事項については，別章を参照のこと．

[2] この最大長制限はソースコード中の `libsent/include/sent/speech.h` で `MAXSPEECHLEN` として定義されている．この値を変えてコンパイルすることで上記の上限を変更可能である．

3.3.1 録音デバイスの条件

動作には，16bit, モノラル (1 チャンネル) で，かつ音響モデルが要求するサンプリングレート（デフォルトは 16kHz）での録音がサポートされている必要がある．

Julius は, OS の入力デバイスの選択やミキサー，録音ボリュームの調整を行わない（一部 OS を除く）．録音のシステム設定は，Julius 外で行う必要がある．

録音デバイスの特性や歪みは音声認識精度に大きな影響を与えるため，事前に録音品質のテストを行うことが望ましい．また，正しく音声を切り出すにはボリューム等を適切に設定する必要がある．Julius に付属の adinrec や adintool は，音声の録音を行えるツールであり，Julius と同一のライブラリを使用しているので，これでファイルに録音するなどして，正しく録音できているかチェックすることが望ましい．また，Julius にオプション `-record` を与えることで，認識した音声データを直接ファイルに保存することができる．

3.3.2 OS ごとの詳細

3.3.2.1 Linux

以下のサウンドインタフェース・ドライバに対応している．

- ALSA ドライバ
- OSS 準拠ドライバ

デフォルト (`-input mic`) では ALSA インタフェースを使用する．`-input oss` / `-input alsa` で明示的に選択することもできる．OSS をデフォルトにしたい場合はコンパイル時に configure に `--with-mictype=oss` をつければよい．

サウンドカードで 16bit, モノラルの録音がサポートされている必要がある．[3] USB オーディオでも動作する．[4]

ALSA のデフォルトの録音デバイス名は `"default"` である．これは環境変数 `ALSADEV` で変更可能である．例えば，複数のサウンドカードが存在する場合，`ALSADEV="plughw:1,0"`のように指定することで，2 枚目のサウンドカードの録音デバイスを指定できる．OSS の場合，デフォルトは `/dev/dsp` であり，環境変数 `AUDIODEV` で変更できる．

3.3.2.2 Windows

Windows では，DirectSound API を使用して音声を録音する．PortAudio ライブラリを使用している．

Portaudio V19 を使用している場合，API は高速な順（ASIO，DirectSound，MME の順）で調べられ，最初に見つかったものが選択される．録音デバイスはを環境変数 `PORTAUDIO_DEV` で明示的に指定することもできる．具体的な指定方法は起動時のログに出力されるので参照のこと．

3.3.2.3 Mac OS

Mac OS X v10.3.9 および v10.4.1 で動作確認されている．CoreAudio API を使用している．

3.3.2.4 FreeBSD

FreeBSD 3.2-RELEASE において，`snd` ドライバで動作確認されている．コンパイル時に自動判別がうまく動かない場合は`--with-mictype=oss` を指定すること．

3.3.2.5 Sun Solaris

デフォルトのデバイス名は `/dev/audio` である．環境変数 `AUDIODEV` で変更できる．Solaris では例外的に Julius はミキサー設定を変更し，録音デバイスがマイクに自動的に切り替わる．

[3] ただし，Linux/OSS では，ステレオ録音しかサポートされていない場合，左チャンネルを認識するよう動作する．

[4] ただし，音声認識で用いるサンプリング周波数（16kHz であることが多い）での録音がデバイス側でサポートされている必要がある．44.1kHz や 48kHz 等の録音のみサポートするデバイスの場合，うまく動作しないことがある．

3.3.3 入力遅延の調整

デバイスからの直接認識では，入力遅延について注意が必要である．通常，PC のオーディオ入力では適度な大きさのデータ片 (chunk / fragment) ごとに処理が行われ，その分の遅延が発生する．Linux (ALSA/OSS) および Windows では，このデータ片の大きさ（＝遅延幅）を環境変数 `LATENCY_MSEC` で指定できる（単位：ミリ秒）．短い値を設定することで入力遅延を小さくできるが，CPU の負荷が大きくなり，また環境によってはプロセスや OS の挙動が不安定になることがある．最適な値は OS やデバイスに大きく依存する．デフォルト値は，Linux では 50 (ミリ秒) である．Windows ではシステムのデフォルト値が用いられるが，動作環境によっては数百ミリ秒になることもある．

3.4 ネットワーク・ソケット経由の入力

3.4.1 ネットワーク経由

`-input adinnet` で，ネットワークソケットから音声データを受け取ることができる．プロトコルは Julius 独自のシンプルなプロトコルを用いており，送受信部は付属のツール adintool に試験的に実装されている．例えば，adintool を用いて以下のようにネットワーク入力が行える．

`Julius` で音声受信・認識：
```
% julius .... -input adinnet -freq srate
```
`adintool` で音声入力・送信：
```
% adintool -in mic -out adinnet -server server_hostname -freq srate
```

なお，Julius 側でサンプリングレートとチャンネル数のチェックは行われないため，クライアント側と Julius 側でサンプリングレートを一致させる必要がある点に注意すること．

3.4.2 esd

Linux では，esd (EsounD daemon) を介して音声を取り込むことができる．esd は多くの Linux デスクトップ環境で利用されている音声デバイス共有用オーディオサーバである．コンパイル時に esd のライブラリ・ヘッダがあれば有効化され，オプション `-input esd` で使用できる．さらに，`--with-mictype=esd` をつけてコンパイルすれば，`-input mic` のデフォルトを esd にできる．

3.4.3 標準入力

標準入力から RAW 形式あるいは WAV 形式の音声入力を受け取ることができる．これには`-input stdin` を指定する．RAW ファイルの場合、サンプリングレートがチェックされないので注意すること．

3.4.4 DATLINK/NetAudio

DATLINK に付属の NetAudio サーバから音声を受け取り認識することができる．これには，コンパイル時に DATLINK/Netaudio のライブラリを組み込んだ上で，`-input netaudio` を指定する．

3.5 特徴量ファイル入力

音声信号ではなく，特徴抽出済みの特徴量ベクトル列を直接 Julius に与えて認識を行うことができる．この場合，Julius は特徴量抽出を行わず，入力のベクトル列はそのまま HMM に照合され，認識が実行される．このため，Julius で抽出がサポートされていない特徴量を持ちいた認識が可能である．

ファイル形式は，HTK 形式の特徴量ファイルをサポートする．`-input htkparam` あるいは `-input mfcfile` を指定する．ファイル名の与え方は音声波形ファイル入力のときと同様である．複数ファイルを連続して認識する場合，音声ファイル入力と同様にオプション `-filelist` を使用する．

特徴量ファイル入力の場合，その特徴量の型が，音響モデルが要求する特徴量の型と一致するかのチェックが行われる．型が完全一致する場合，そのまま認識を行う．一致しない場合，ベースフォームが同じでかつ以下のいずれかの処理を入力に行えば一致させられる場合は，その処理を行ってから認識を行う．

- 一次差分 (_D) の追加あるいは削除
- 二次差分 (_A) の追加あるいは削除
- エネルギー項の抑圧 (_N)

上記の処理によっても一致させられない場合，あるいはベースフォームが異なる場合は，エラーを出力してそのファイルの処理をスキップする．なお，このパラメータチェックは，オプション `-notypecheck` を指定すれば無効化できる．

3.6 プラグインによる入力拡張について

上記までは Julius に組み込まれている音声入力方法について説明したが，4.1 以降では，外部プラグインによって新たに入力を拡張することができる．プラグインでは，音声波形信号入力のほか，特徴量ベクトル列のリアルタイム入力もサポートする．プラグインを作成することで，対応する音声デバイスを拡張したり，ネットワークから特徴量を受け取るようなことが可能となる．詳細は，プラグインの章を参考にされたい．

Chapter 4

フロントエンド処理・特徴量抽出

Julius は，入力音声から音声認識のための音声特徴量を抽出する．音声特徴量は，短時間ごとに切り出された音声信号から抽出される特徴ベクトルの時系列であり，特徴抽出後，得られた特徴量列に対して認識処理（解探索）を行う．また，Julius には特徴抽出の前処理としていくつかのフロントエンド処理が実装されている．プラグインによって処理を追加することもできる．この章では，Julius の特徴量抽出を含むフロントエンド処理，および設定方法について説明する．

4.1 フロントエンド処理

フロントエンド処理は，入力音声波形に対する最初の信号処理段階である．これらは，特徴量抽出の前に行われる．以下に使用可能な処理を列挙する．

4.1.1 直流成分除去

特徴量抽出の前段階として，直流成分除去を行える．直流成分であるオフセット値の推定方法として，短時間音声区間（フレーム）ごとに行う方法（オプション`-zmeanframe`）と，長時間平均を用いる方法（オプション`-zmean`）の二とおりが用意されている．前者は，特徴抽出の直前において，切り出された短時間フレームごとにオフセット推定と除去を行う．後者は，Julius 起動後，無音を含めた最初の 48,000 サンプル分の振幅平均を直流成分（オフセット）として以降の入力の直流成分除去を行う．`-zmeanframe` は HTK の `ZMEANSOURCE` と同じ処理である．

オプション指定時の注意：前者の `-zmeanframe` は，特徴量抽出のオプションであり，音響モデルごとに指定する必要がある．特に，複数の音響モデルを使用する場合，音響モデルごとに個別に指定する必要がある．一方，後者の `-zmean` はグローバルオプションであり，1 回指定するだけでよい．

4.1.2 スペクトルサブトラクション

雑音のスペクトルを推定して音声信号から減算することで雑音の影響を抑圧する，スペクトルサブトラクションを行うことができる．これは，特にファン音などの定常雑音の除去に一定の効果を持つ．

雑音スペクトルの推定方法としては，(1) 各入力の最初の数百ミリ秒を雑音区間と仮定してその平均を雑音スペクトルとする（オプション`-sscalc`, `-sscalclen`），(2) あらかじめ付属のツール mkss で雑音スペクトルを推定してファイルに保存しておき，それを `-ssload` で読み込む方法 の 2 種類がある．音声ファイルに対する入力ではどちらかを指定できるが，マイク等のオンライン入力では後者のみ使える．

減算の大きさは `-ssalpha` で指定できる．雑音スペクトルのパワーにこの係数をかけた値が入力音声のスペクトルから差し引かれる．また，減算後に負になってしまう帯域については，減算を行う代わりに，入力スペクトルがフロアリング係数（`-ssfloor`）倍される．

なお，スペクトルサブトラクションは特徴量抽出段階で行われるため，音響モデルごとに指定する必要がある．複数音響モデルを使用する場合は音響モデルごとに個別に指定する必要があるので注意すること．

4.2 特徴量抽出

4.2.1 サポートする特徴量

Julius で抽出できる特徴量は，メル周波数ケプストラム係数 (mel-frequency cepstral coefficient; MFCC) に基づく特徴量である．具体的には，HTK で抽出できる MFCC のバリエーションを網羅しており，MFCC およ

びその派生パラメータ（パワー，0 次ケプストラム，およびそれらの一次差分と二次差分）を抽出できる．また，正規化法として，ケプストラム平均正規化 (cepstral mean normalization; CMN)，パワー項正規化，ケプストラム分散正規化 (cepstral variance normalization; CVN) をサポートする．また，ワーピング係数を与えることでいわゆる声道長正規化 (vocal tract length normalization; VTLN) も行える．

指定できるパラメータは HTK に準拠している。基本パラメータ種 (basic parameter kind: basekind) は MFCC のみサポートする。修飾子は全ての修飾子 (_N, _D, _A, _Z, _0) をサポートする。また、CRC チェックサム付き_K や圧縮済み_C の特徴量ファイルも読み込める。パワー項と_0 が両方指定されている場合、それらは両方とも特徴量ベクトルに含まれる。

音声波形を直接認識できるのは，音響モデルが以上で述べたような MFCC をベースとする特徴量で学習されている場合のみである．Julius は MFCC 以外の音響モデルも扱うことができるが，その場合，波形ファイルやマイクなどの音声入力を直接認識することはできない．代わりに，HTK 形式の特徴量パラメータファイルのみを入力とすることができる (`-input mfcfile`)．

複数の音響モデルを用いる場合，特徴量抽出条件は使用する音響モデルごとに独立して設定する必要がある．同一の特徴量を用いる場合でも，それぞれの音響モデルごとに繰り返し指定する必要がある点に注意されたい．特に，サンプリングレート・窓幅・窓シフト長は全ての音響モデルで値が一致している必要がある（一致していなければ Julius はエラー終了する）．

4.2.2 特徴量抽出パラメータの指定方法

認識時には，特徴量の型以外にも，フレーム窓幅やシフト幅，フィルタバンク数，各種正規化の有無などの抽出条件を，モデル学習時と正確に一致させる必要がある．特徴量の抽出条件や各種パラメータの設定方法は，オプションで値を直接指定するほかに，HTK で学習時に使った Config ファイルを直接読み込んだり，音響モデルを Julius 用のバイナリ音響モデルへ返還するときにヘッダ内に条件を埋め込んだりできる．各設定方法の詳細を以下に述べる．

4.2.2.1 方法１：直接指定

特徴量抽出条件の設定オプション，および HTK の設定との対応を，表 4.1 に示す．

表の最初の 2 項目「パラメータ種」および「ケプストラム次数」は，使用する音響モデルのヘッダ情報から自動的に取得されるため，明示的に指定する必要はない．

特徴量の設定のほとんどは，HTK と互換性がある．以下に，HTK との違いついて注意すべき点を列挙する．

- デフォルト値に "(*)" がついているものは，HTK のデフォルト値と異なる．Julius のデフォルト値は，一緒に配布されている標準音響モデルの設定をデフォルトとしている．
- 対応する HTK オプションに "(*)" がついているものは，指定時の値のユニットが HTK と異なるので注意すること．
- 窓関数はハミング窓に固定されている．

4.2.2.2 方法２：HTK Config の読み込み

方法 1 のように個々のパラメータを指定する代わりに，HTK の Config ファイルを直接読みこませることができる．オプション `-htkconf` で HTK Config ファイルを与えると，Julius は与えられたファイルを解析し，対応する設定を見つけるとその設定値を Julius の設定値に変換してセットする．なお，表中に無い設定項目は無視される．

この方法 2 を用いる場合，Config ファイル内で明示的に指定されていないパラメータのデフォルト値は，Julius 独自のものではなく HTK のデフォルト値となる．

4.2.2.3 方法３：バイナリ HMM ファイルへのパラメータ埋め込み

Julius の付属ツール mkbinhmm で，HTK ascii 形式の音響モデルを Julius 用バイナリ HMM ファイルへ変換できるが，その際に，出力ファイル内に特徴量抽出条件パラメータを埋め込むことが可能である．モデルに必要なパラメータをあらかじめ埋め込んでおけば，モデルを読み込む際に同時にそのモデルで必要な特徴量抽出条件もセットすることができ，モデルと別個にオプションを指定する必要がないメリットがある．

埋め込み方法は，mkbinhmm で Julius バイナリ形式へ変換する際に，mkbinhmm に対して Julius と同じ特徴量オプションを与える．与え方は，Julius と同様に，前述の 1 あるいは 2 の方法が使える．なお，特徴量に関係ないパラメータは無視されるので，Julius の認識時に使っているすべてのパラメータをそのまま mkbinhmm に与えればよい．

Table 4.1 特徴量抽出条件の設定オプションとデフォルト値

オプション	内容	デフォルト値	対応する HTK オプション
	パラメータ種	"MFCC"	TARGETKIND
	ケプストラム次数	-	NUMCEPS
-smpFreq Hz	入力波形のサンプリングレート	16000	SOURCERATE (*)
-fsize samples	ウィンドウ幅	400 (=25ms in 16kHz)	WINDOWSIZE (*)
-fshift samples	ウィンドウシフト長	160 (=10ms in 16kHz)	TARGETRATE (*)
-zmeanframe	フレーム単位の直流成分除去	no	ZMEANSOURCE
-preemph value	高域強調係数	0.97	PREEMCOEF
-fbank value	フィルタバンクチャンネル数	24 (*)	NUMCHANS
-ceplif value	ケプストラムのリフタリング係数	22	CEPLIFTER
-delwin frames	一次差分計算用窓幅	2	DELTAWINDOW
-accwin frames	二次差分計算用窓幅	2	ACCWINDOW
-lofreq Hz	周波数カットオフの下限	-1 (disable)	LOFREQ
-hifreq Hz	周波数カットオフの上限	-1 (disable)	HIFREQ
-rawe / -norawe	高域強調前のエネルギーを使用	no (*)	RAWENERGY
-enormal / -noenormal	対数エネルギー項を正規化	no (*)	ENORMALISE
-escale value	対数エネルギー正規化のスケーリング係数	1.0 (disable) (*)	ESCALE
-silfloor dB	対数エネルギー正規化のフロアリング係数	50.0	SILFLOOR
-usepower	フィルタバンク解析で振幅の代わりにパワーを使う (rev.4.0.2)	no	USEPOWER
-cvn / -nocvn	ケプストラム分散正規化 (rev.4.0.2)	no	-
-vtln alpha flo fhi	声道長正規化 (rev.4.0.2)	1.0 any any (disable)	WARPFREQ,WARPLCUTOFF,WARPHC
パラメータ種，ケプストラム次数は音響モデルによって与えられる．デフォルト値の (*) は HTK のデフォルト値と異なる．また対応オプションの (*) は HTK と指定する数のユニットが異なる．			

```
% mkbinhmm -htkconf ConfigFile -C ... hmmdefs binhmm
```

また，既にあるバイナリ音響モデルに対してパラメータ情報を追加・上書きするには，以下のようにバイナリ音響モデルを入力として与えればよい．

```
% mkbinhmm -htkconf ConfigFile -C ... binhmm1 binhmm2
```

この方法は，バージョン 3.5.3 以降でサポートされている．この方法を用いることで，音響モデルの作成者は，そのモデルを用いた認識の際に必要な特徴量情報をモデル内に埋め込んでおくことができ，利用者側で設定する必要がなくなる．特に音響モデルを配布する場合などに推奨される方法である．

なお，複数の方法でパラメータが指定された場合，(1) オプション指定 (2) `-htkconf` の指定 (3) バイナリ音響モデルに埋め込まれた設定 (4) デフォルト値，の順で (1) が最も優先される．

4.3 正規化処理

環境や話者の影響を軽減するため，算出後の特徴量に対して正規化処理を行うことができる．以下に可能な処理を述べる．

4.3.1 ケプストラム平均正規化 (CMN)

長時間のケプストラム平均を差し引くケプストラム平均正規化 (cepstral mean normalization; CMN) を行うことができる．使用する音響モデルが CMN 付きの MFCC で学習されている場合（すなわち MFCC 型定義に"_Z" が含まれているとき），CMN が自動的に有効となる．

ファイル入力では，CMN は発話単位で行われる．すなわち，その入力発話自身のケプストラム平均を求めたあと CMN が実行され，認識処理が行われる．マイクなどのリアルタイム入力では，発話全体の特徴量が得られないため，代わりに後述の MAP-CMN が用いられる．このため，マイクなどの直接認識時とファイル入力時では認識結果が異なる場合がある点に注意されたい．

4.3.2 ケプストラム分散正規化 (CVN)

CMN に加えて，ケプストラムの分散が 1 になるよう正規化を行うケプストラム分散正規化 (cepstral variance normalization; CVN) を行うことができる．オプション`-cvn` で有効化される．処理単位や直接認識時の注意点は CMN と同じである．HTK の CVN の実装とは分散推定の単位が異なるので同一ではない．

4.3.3 周波数ワーピング（声道長正規化）

オプション `-vtln` で周波数ワーピングを行うことができる．オプションにつづけてワーピング係数，下端周波数，上端周波数を指定する．あるいは，HTK Config 読み込み時にワーピングに関する設定がある場合にも有効化される．この機能は，主に声道長正規化 (vocal tract length normalization; VTLN) のために用いられるものである．なお，Julius 自身には，VTLN のための周波数ワーピング係数のパラメータ推定は実装されていない．

4.4 リアルタイム認識における正規化

マイクやネットワーク入力では，入力と認識処理が並行して行われるが，この場合，その発話全体の情報に基づく特徴量正規化が行えない．以下，リアルタイム認識における正規化処理について説明する．

4.4.1 実時間エネルギー正規化

特徴量の中にエネルギー項がある場合（すなわち MFCC 型定義に"_E" があり "_N" がないとき），発話全体の平均で正規化することがある．Julius では，オプション`-enormal` を指定することでこのエネルギー項正規化を行える．

Julius では，音響モデルの特徴量タイプにこの実時間エネルギー正規化が指定されている場合，リアルタイム認識では，直前の発話のエネルギー平均を用いて正規化が行われる．ただし，起動して最初の発話は正規化が行われず，また直前の入力から入力環境が大きく変化すると追従できないことがあるので注意すること．[1]

[1] 基本的に，リアルタイム認識用の音響モデルでは，正規化エネルギー項は使わないほうがよい．

4.4.2 実時間ケプストラム平均・分散正規化

リアルタイム認識では，通常の CMN の代わりに MAP-CMN が行われる．MAP-CMN では，入力開始時はあらかじめ与えられているケプストラム平均を初期値として CMN をスタートし，入力が進むにつれて徐々にその発話自身のケプストラム平均を使う．なお，オプションでこの MAP-CMN の振る舞いを制御したり，初期値をあらかじめ与えることができる．通常，初期値としては，直前の発話の平均が用いられる．

また，分散正規化（CVN）が有効化されているときは，CVN に対しても上記と同様の方法で処理が行われる．

4.4.3 プラグインによる拡張について

プラグインでフロントエンド処理を追加できる．入力ストリーム全体，あるいは零交差によるトリガ部分の音声データに対して処理を行うプラグイン，および特徴量ベクトルに対して（認識前に）処理をかけるプラグインを記述できる．また，音声入力や特徴量入力のデバイスをプラグインとして追加することもできる．詳しくはプラグインの章を参考のこと．

Chapter 5

音声区間検出・入力棄却

Julius では，入力音声ストリームから音声が発話された区間を検出する音声区間検出（voice activity detection: VAD）の機構をいくつか持っている．デフォルトでは，音声波形の振幅と零交差に基づく入力検知のみを行う．この方法は雑音の大きい（SNR の低い）環境ではうまく動作しない傾向にあるため，より頑健な方法として，混合ガウス分布モデル (Gaussian mixture model; GMM) に基づくフレーム単位の音声・非音声識別に基づく区間検出，およびデコーダの認識中の仮説情報をもとに区間検出を行うデコーダベース VAD が実装されている．また，音声認識処理の後処理として，入力区間全体を認識した後に，入力の棄却を判定する入力棄却も備えている．

これらの方法は，マイク入力のみならず音声ファイル入力に対しても行うことができる．

以下，Julius における音声区間検出および入力棄却について述べる．

5.1 音声区間検出

一般に，音声区間検出は，入力ストリームに対して短時間ごとの特徴から音声区間の開始・終了を検出し，それを元に認識単位の切り出しおよび発話単位の区切りを行う方法である．以下，Julius で用いている方法について解説する．

5.1.1 振幅と零交差に基づく入力検知

音声信号の振幅と零交差数に基づいて，音入力の開始と終了を検出する方法である．一定のレベルを越える振幅について零交差数が一定数を越えたとき，音声始端として認識処理を開始する．そして，値が一定以下になったときに，音声の終端としてそこで入力を区切って 1 発話分の認識を終了する．

この方法は最も基本的な方法であり，計算量が少ないという利点がある．ただし，入力の振幅は実行環境（録音ボリューム，マイクと発話者の距離等）に大きく左右され，実行環境ごとにしきい値の調整が必要である．また，SN 比が低い環境では性能が著しく悪化する．

この振幅と零交差に基づく入力検知は，デフォルトではマイクロフォン等の直接入力に対して ON, ファイル入力に対して OFF となっている．これはオプション `-cutsilence` で ON に，`-nocutsilence` で OFF に明示的に切り替えできる．ファイル入力では，通常は 1 ファイル全体を 1 つの発話として認識が行われるが，ON にすることでマイクと同様の切り出しを行うこともできる．

振幅の検出レベルは`-lv`, 零交差数は `-zc` で指定する．切り出しの性能は，これらのしきい値に大きく依存する．

また，Julius では開始部や末尾の減衰部分を区間に確実に含めるために，検出した区間の前後にマージンを設けて切り出している．区間前後のマージン長はそれぞれ`-headmargin`, `-tailmargin` で指定できる．

5.1.2 GMM による音声区間検出

ガウス混合分布モデル (Gaussian mixture model; GMM) に基づく音声区間検出が行える．音声と非音声の GMM を定義し，入力短時間フレームごとに特徴量抽出から各 GMM の尤度計算を行い，音声 GMM と非音声 GMM の尤度比から，音声区間の開始・終了を判別する．

この試験的な機能は，デフォルトでは無効化されており，ソースからコンパイルする際に `configure` に`--enable-gmm-vad` を指定することで有効となる．なお，有効にした場合，あとで述べる GMM に基づく入力棄却は無効となる．

この処理は，前節の，振幅と零交差によって検知された音声波形に対して後処理として行われる．

GMM は，3 状態（出力状態が 1 つのみ）の HMM として，HTK 形式の定義ファイルとして作成し，これを -gmm で与える．モデル内では音声の GMM, 非音声の GMM をそれぞれ複数定義できる．棄却したい非音声 GMM の名前を -gmmreject で与える（複数ある場合はカンマで区切って与える）

GMM 用の特徴量抽出パラメータは，同時に使用する音響モデルと異なる特徴量を別途指定することも可能であり，同じ特徴量ベクトルを共有することもできる．別途指定する場合は，インスタンス宣言オプション-AM_GMM のあとに音響モデルと同じ要領で指定する．音響モデルと同様の任意の特徴量設定ができるが，サンプリング周波数，フレームサイズ，フレームシフト幅の 3 つのみ，同時に使う音響モデルと同一である必要がある（同一でない場合エラーとなる）．なお，-AM_GMM の指定がない場合は，GMM の特徴量は，最初に指定された音響モデルと同じであると仮定して処理される．

5.1.3 デコーダベースの音声区間検出

実験的機能として，認識途中の部分文仮説の状態に基づいて認識の中断や再開，無音区間のスキップを行うデコーダベースの音声区間検出が行える．認識処理を常に行いながら，有効な候補が現れた区間のみ認識結果を出力する．手法についての詳細は，探索アルゴリズムの章の「認識処理に基づく入力区切り」の節を参照のこと．

デコーダに基づく VAD は，デフォルトでは無効化されており，使用するにはコンパイル時に configure オプションで--enable-decoder-vad を指定し，実行時に-spsegment を指定する．

5.2 入力棄却

入力処理の終了後に，事後的に入力を棄却することができる．認識後の後処理として，その区間全体の第 1 パスの認識情報や認識結果をもとに，その入力の棄却を判定する．棄却時，第 2 パスは実行されない．

5.2.1 GMM に基づく入力棄却

GMM の入力区間全体での累積尤度から，入力棄却が行える．入力終了後にその入力区間全体がどのモデルに最も近いかを識別し，非音声だった場合に，その入力を棄却する．

GMM は，3 状態（出力状態が 1 つのみ）の HMM として定義する．形式は HTK 形式で与える．形式や使用できる特徴量の制限は音響モデルと同じである．GMM で用いる MFCC 特徴量の設定は，-AM_GMM のあとに音響モデルと同様に指定する．この特徴量設定は音響モデルと別に，明示的に指定する必要があることに注意が必要である.（アルゴリズムの解説）

5.2.2 平均パワーによる棄却

この試験的な機能は，デフォルトでは無効化されており，ソースからコンパイルする際に configure に--enable-power-reject を指定することで有効となる．リアルタイム認識時のみ用いることが出来る．なお，特徴量抽出でパワー項を用いていない場合は使用できない．

Chapter 6

音響モデル

Julius は音響モデルとして，HMM (Hidden Markov Model) を用いることができる．コンテキスト非依存モデル (monophone)，およびコンテキスト依存モデルを triphone まで扱える．出力確率モデルは，対角共分散のガウス混合分布を基本とし，tied-mixture モデルや phonetic tied-mixture モデルも扱える．本章では Julius で扱うことのできる音響 HMM の仕様を述べるとともに，論理音素名から物理音素名へのマッピングや単語間トライフォンの扱いなどについても解説する．

6.1 モデルの仕様

Julius は，以下のような HMM を扱うことができる．

- MFCC ベースの特徴量で学習されたもの（MFCC 以外の場合，音声波形を直接認識することはできず，特徴抽出済み特徴量ファイルのみ入力とできる）
- 音素のコンテキスト依存性は，トライフォンに対応
- マルチストリームに対応（バージョン 4.1 以降, それ以前はシングルストリームのみ）.
- 確率分布は，連続分布のみ．離散は対応していない．
- 共分散行列の型は，対角共分散のみ．
- 継続時間長モデルは，対応していない．
- パラメータ共有は，モデル，遷移行列，状態，ガウス分布，共分散それぞれのレベルで行える．
- Tied-mixture モデルに対応
- Multi-Space Probability Distribution HMM (MSD-HMM) に対応 (configure --enable-msd 時)．HTS で学習された音響モデルを扱える．（バージョン 4.1 以降）
- 話者適応は，サポートされていない．
- モデル名の長さの上限は 256 文字，最大ストリーム数は 50，状態数の制限は無い．

音響モデルが triphone であるかどうかは，内部の定義モデル名から自動判別される．モデル名が ’+’ や ’−’ を含んでいると，triphone モデルであると認識される．

トライフォン使用時は，同時に HMMList ファイルも指定する必要がある．HMMlist ファイルは，辞書上の音素表記と音響モデル内の HMM 名のマッピングを指定するものである．特にトライフォン使用時は，登場しうる全てのトライフォンに対して，対応する定義 HMM 名へのマッピングを記述した HMMList を与える必要がある．HMMList ファイルの仕様は本章の別節を参照のこと．

6.2 音響モデルのファイル形式

6.2.1 HTK ascii 形式

Julius は HTK の ASCII 形式の HMM 定義ファイル (MMF) を読み込むことができる（HTK のバイナリ形式を読み込むことはできない）．全ての音素定義を 1 つにまとめた単一の MMF ファイル (`hmmdefs`) として与えることができる．[1] ファイルは gzip で圧縮されたものをそのまま読み込める．

使用する際には，オプション "`-h`" で指定する．

共有マクロは `t` (遷移)，`s` (状態)，`m` (分布)，`h` (モデル)，`v` (共分散) を扱える．4.1 以降では，さらに `w` (ストリーム重み)，`p` (混合分布マクロ: HTS 拡張) を扱える．これ以外の共有マクロ (`u i x` 等) には現バージョンでは対応していない．

定義ファイル中に "<TMix>" が含まれている場合，Julius はそのモデルを Tied-mixture モデルとして扱う．この場合，認識中の音響尤度の計算順序が tied-mixture 用のものに変更される（計算単位が状態単位からコードブック単位に変更される）．単一のコードブックからなる通常の tied-mixture モデルのほか，複数のコードブックを扱うこともできる．このため，例えば音素ごとにコードブックを持つ phonetic tied-mixture のようなモデルも扱える．

Julius は音響モデル適応をサポートしていない．音響モデル定義内に適応用の regression tree がある場合，それらは無視される．

また，モデルの定義においては，各モデルの初期状態と最終状態は出力分布を持ってはならない[2]．状態遷移については，すべての状態遷移を扱うことができる．初期状態から，あるいは最終状態への遷移がそれぞれ複数存在する場合，Julius はマルチパスモードで実行される（後述）．

音声波形を直接認識したい場合は，HMM が MFCC をベースとする特徴量で学習されている必要がある．特徴量の詳細については，特徴量抽出の章を参照されたい．

6.2.2 Julius 用バイナリ形式

Julius は HTK の ASCII 形式の HMM 定義ファイルを読み込めるが，これはテキスト形式であり，モデル構造を解析しながら読み込むため，大規模なモデルでは読み込みに時間がかかる．あらかじめ Julius 用バイナリ形式へ変換しておくことで，読み込み時間を短縮できる．また，gzip で圧縮されたファイルもそのまま読み込める．

Julius 用バイナリ形式への変換は，付属のツール mkbinhmm を用いる．使用例を以下に示す．

```
% mkbinhmm hmmdefs binhmm
```

また，バイナリ形式では，その音響モデルで認識時に必要とされる特徴量抽出の条件パラメータを埋め込むことができる．これによって，音響モデルごとにパラメータを直接設定する必要がなくなる．オプションは，Julius で認識で使用する設定ファイルをそのまま mkbinhmm に与えればよい（無関係なオプション指定は無視される）．以下に例を示す．

```
% mkbinhmm -htkconf ConfigFile -C ... hmmdefs binhmm
```

これによって，hmmdefs がバイナリ形式の `binhmm` へ変換されると同時に，ヘッダ内に特徴量パラメータが書き込まれる．Julius で使用する際には，このバイナリ音響モデルを指定するだけで，必要な特徴量パラメータ設定がされる．詳細は，特徴量抽出の章のセクション「方法 3：バイナリ HMM ファイルへ埋め込む」を参照のこと．

使用する際には，ASCII 形式と同様に，オプション "`-h`" で指定する．ファイル形式は自動判別される．

6.3 HMMList ファイル

Julius では，音響モデルとともに HMMList と呼ばれるファイルで，単語辞書の音素表記，あるいはそこから生成されるトライフォンの論理音素名と実際の音響モデル上の物理音素名との対応を与えることができる．特に，トライフォン使用時には必須である．

トライフォン使用時，HMMList には，辞書上の単語内および単語間で登場しうる，すべてのトライフォンを記述する必要がある．対応が与えられていないトライフォンが起動時あるいは認識処理中に出現した場合，エラーとなる．

使用する際にはオプション "`-hlist`" で指定する．なお，ファイル形式は自動判別される．

[1] 音素ごとに別々の定義ファイルとして与えることはできない．
[2] これは HTK と同じ制限である．

6.3.1 テキスト形式

標準でサポートされているファイル形式は，テキスト形式である．フォーマットは以下のとおり．

- 一行に 1 つの対応を定義する．
- 第 1 カラムに論理音素名，第 2 カラムに対応する物理音素名（hmmdefs 内の HMM 名）を指定する．
- 第 2 カラムを省略した場合，第 1 カラムと同じと解釈される．
- 同じ論理音素名を重複して登録した場合，エラーとなる．

以下に例を示す．第 2 カラムが空白のエントリは，その HMM 名が直接音響モデル内で定義されていることを意味する．

```
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
```

6.3.2 バイナリ形式

バージョン 4.1 より，バイナリ形式がサポートされている．バイナリ形式では，検索用のパトリシア木インデックスも同時に保存されるため，読み込み時に検索インデックスを生成する必要がなく，起動が高速化される．

テキスト形式からバイナリ形式への変換は，付属のツール mkbinhmmlist を使用する．以下は，HMMList ファイル `hmmlist` をバイナリ形式に変換して `binhmmlist` に保存する例である．

```
% mkbinhmmlist hmmdefs hmmlist binhmmlist
```

変換の際は，定義のチェックのため，一緒に使用する予定の音響モデル定義ファイル *`hmmdefs`* も与える必要がある．なお，この音響モデルのファイル形式は HTK ASCII 形式 / Julius バイナリ形式のどちらでも構わない．

6.4 音素コンテキスト依存モデル

Julius における音素コンテキスト依存モデルの扱いについて述べる．Julius は，与えられた音韻モデルが音素環境依存モデルであるかどうかを，HMM の名称パターンから判定する．HMM 名の中に記号 ‘+’, ‘−’ が両方とも 1 つ以上含まれていれば，そのモデルは音素環境依存モデルとして扱われる．なお，依存関係は前後 1 履歴（トライフォン）までのみ考慮される．

トライフォンを使う場合，認識アルゴリズムも音素環境依存性を考慮したものになる．辞書の読み込み時に，単語内の音素をトライフォンに変換しながら読み込まれる．単語間の依存性は，第 1 パスでは近似計算が行われ，第 2 パスで正確に計算される．

6.4.1 論理音素名と物理音素名

音素環境依存モデルを使用する場合，単語辞書上の音素列をトライフォンに変換して認識を行う．辞書上のモノフォン表記からトライフォンへの変換は，ある音素 "X" に対して直前の音素が "L"，直後の音素が "R" である場合に"L-X+R" の形で行われる．たとえば，以下のような単語「英訳」を考える．

```
英訳+エイヤク+17        [英訳]  e     i     y     a     k     u
```

このトライフォン表記への変換は以下のようになる．なお単語の先頭・末尾については特別な処理が行われる（詳細は後述）．

```
英訳+エイヤク+17        [英訳]  e+i e-i+y i-y+a y-a+k a-k+u k-u
```

Julius では，このような，単語辞書上の音素表記およびそれから生成されるモデル参照名を「論理音素名 (logical phone)」，それに対して実際に音響モデル上で定義されている HMM 名を「物理音素名 (physical phone)」と呼ぶ．

論理音素名と同一名の物理音素名のモデルが音響モデル上で定義されていればよいが，実際には，学習時のコンテキストクラスタリングにより，辞書上のすべての可能なトライフォンについて HMM が定義されていないことがある．また，全ての単語間の接続を考えるとき，音響モデル学習時に出現しなかった音素並びが登場しうる．このため，音素環境依存モデルを用いる場合は，登場する可能性のある全ての論理音素名について，対応する物理音素名へのマッピングを HMMList ファイルとして与える必要がある．これについて次節で述べる．

6.4.2 HMMList ファイルによるマッピング

HMMList ファイルは，登場しうるすべてのトライフォン論理音素名に対して，音響モデルで定義されている物理音素名へのマッピングを指定する．トライフォン使用時には必ず指定する必要がある．ファイル形式については前節を参照のこと．

HMMList ファイルでのマッピング指定は，hmmdefs 内の定義に優先するので注意すること．例えば，hmmdefs において

```
~h "j-u+q"
```

という HMM が定義されているとき，HMMList ファイルで

```
j-u+q y-u+q
```

のように指定すると，`j-u+q` の HMM は実際には使用されず，`y-u+q` が使われることになる．

システムにおいて実際にどのようにマッピングされているかは，julius を`-check triphone` を付けて起動することでチェックできる．プロンプトに対して `"H"` と入力すればヘルプが出てくる．

6.4.3 単語間トライフォン近似：pseudo phone

トライフォンの場合，連続音声認識において，ある単語内の先頭と末尾の音素は，その単語の前後に接続する単語の読みの影響を受けて変化する．このことは，ある単語の尤度計算を，その前後の単語に依存して分けて行う必要があることを意味し，大語彙では計算量が増大する．この単語をまたぐ音素環境依存性（単語間トライフォン）に対して，Julius では第 1 パスで "pseudo phone" を用いた近似計算を行い，第 2 パスで正確なトライフォンを計算する方法を用いている．

Pseudo phone は，中心音素が同じ論理音素名を持つトライフォン HMM の集合である．例えば，読みが `"k u"` で終わる単語が辞書にあるとき，その末尾の音素 `"u"` に対して，`"k-u+i"`, `"k-u+b"`, `"k-u+m"` などの左コンテキストと中心音素が`"k-u"` であるトライフォンの集合を抽出し，その集合を `k-u` のような擬似的なバイフォンとして割りつける．これを Pseudo phone と呼ぶ．上記の `k-u(+*)` のように同じ中心音素と左コンテキストを持つトライフォンの集合，および `(*-)b+e` のように同じ中心音素と右コンテキストを持つ集合を pseudo biphone，`(*-)a(+*)` のように中心音素が同じである全てのトライフォン集合は pseudo monophone と呼ばれる．

Pseudo phone のリストは，音響モデルおよび HMMList ファイルを読み込んだあとに，HMMList にある論理トライフォン名から自動的に生成される．

認識時には，pseudo phone は，それ自身が一つの論理バイフォンあるいは論理モノフォンとして辞書上の単語の先頭および末尾に配置される．第 1 パスではその集合内のすべてのトライフォンについて尤度計算を行い，スコアの高いものがその pseudo phone のスコアとして与えられる．第 2 パスでは，単語仮説のコンテキストから正確なトライフォンに置き換えられて尤度計算される．

なお，HMMList でバイフォン表記から物理音素名への明示的なマッピングが記述されている場合，そのマッピングは pseudo phone より優先される．HMMList 中にバイフォンやモノフォンが明示的に指定されている場合は，pseudo phone は使用されず，指定されたモデルが使用される．

6.5 状態間遷移とマルチパスモード

音素間のパスの扱いの効率化のために，Julius は，与えられた音響モデル内の状態遷移のパターンを調べ，そのパターンに対応する効率的な計算処理を選択する．以下に具体的に説明する．

音響モデル内の全ての HMM が以下の条件を全て満たすとき，モデル間の遷移がただ 1 つとなるので，Julius は処理を簡素化した高速なモードで動作する．これが Julius の通常モードである．

- 初期状態からの遷移が 1 つのみ

- 最終状態への遷移が１つのみ

上記の条件に当てはまらないモデルの場合，通常モードではうまく扱えないため，Julius は「マルチパスモード」で動作する．マルチパスモードでは，任意の状態間遷移を扱えるよう拡張されたアルゴリズムが使用される．このモードでは，初期状態から最終状態への直接遷移（すなわちモデルそのものをスキップする遷移）も扱える．

マルチパスモードでは，音素間処理が複雑になるため認識速度が通常モードよりも 1,2 割程度低下する．また，木構造化辞書において単語の先頭および末尾に対応するノードが新たに作られるため，木構造化辞書が大きくなり，ビーム幅を通常よりも若干広めに取る必要がある．

通常モードとマルチパスモードの切り替えは，与えられた音響モデルの遷移パターンから自動的に行われるほか，オプション `-multipath`, `-nomultipath` で明示的に指定することもできる．また，複数の音響モデルを用いる場合，モードは個々の音響モデルに対して設定できる．

6.6 複数音響モデルによるマルチデコーディング時の注意

複数の音響モデルを用いる場合，音響モデルごとに特徴量抽出のパラメータを個別に設定する必要がある．モデル間で同一の特徴量を用いたい場合は，音響モデルごとに繰り返し同じ特徴量パラメータを指定する必要があることに注意すること．また，音声信号のサンプリング周波数，窓サイズ長，およびフレームシフト幅についても，音響モデル間で同一の値を設定する必要がある．

Chapter 7

言語モデル

Julius は認識のための言語制約として，いくつかの種類の言語モデルをサポートしている．統計モデルである単語 N-gram モデルに基づく認識，記述文法に基づく認識，および単語リストのみによる孤立単語認識を行うことができる．

また，ライブラリとして他のアプリケーションに組み込まれるとき，アプリケーション側から何らかの言語制約を提供するユーザ関数を与えることで，その言語制約を直接駆動して認識処理を行うことができる．

言語モデルは，言語モデルインスタンスごとに個別に指定できる．一つの言語モデルインスタンスには一つのモデルしか指定できない（ただし文法については一つのインスタンス内で複数の文法を使うことができる）．複数の言語モデルインスタンスに対してそれぞれ異なるタイプのモデルを指定することで，複数の異なるモデルを同時に用いて認識することもできる．

本章では，各言語モデルで共通である単語辞書について最初に述べた後，各種言語モデルについてファイル形式や構築方法，Julius への指定方法，制限等について述べる．

7.1 単語辞書

単語辞書は，認識対象とする単語とその読み（音素列表記）を定義する．Julius は，この単語を最小ユニットとして解探索を行う．

認識用辞書のフォーマットは言語モデル間で共通である．ただし第 1 フィールドの言語エントリは言語モデルによって扱いが異なり，第 2 フィールドは N-gram でのみ有効である．文法の場合は，認識用文法の語彙ファイル (.voca) からコンパイラによって認識用辞書を生成して使用する．

Julius に単語辞書を与える方法は，言語モデルごとに異なる．以降に続く，各言語モデルの説明の節を参照のこと．

7.1.1 ファイル形式

単語辞書のファイル形式は，HTK の辞書フォーマットを拡張したものであり，一行に一つずつ，単語とその読みを定義する．各フィールドは空白またはタブで区切られる．以下に詳細を述べる．

7.1.1.1 第 1 フィールド：言語エントリ（必須）

第 1 フィールドでは，その単語の言語制約の対応エントリを書く．単語 N-gram では N-gram 上の語彙，文法では属するカテゴリ番号となる．

7.1.1.2 第 2 フィールド：エントリ内確率（オプション）

単語 N-gram の場合，‘@’ に続けて確率を常用対数 (log10) で記述することで，言語エントリ内の生起確率を追加指定することができる．以下に例を示す．

```
＜地名＞ @-2.33251 京都+52 [京都] ky o: t o
＜地名＞ @-1.68893 奈良+52 [奈良] n a r a
＜地名＞ @-2.63574 和歌山+52 [和歌山] w a k a y a m a
```

この例では，各単語の出現確率は「＜地名＞」という単語の N-gram 上の出現確率（対数尤度）にエントリ内生起確率を足した値となる．

Julius では，このように，N-gram としてクラス単位の N-gram を与え，認識用辞書で，単語ごとにその属するクラスとクラス内確率を上記の要領で記述することで，クラス N-gram が実現できる．

7.1.1.3 第 3 フィールド：出力文字列（オプション）

その単語が認識されたときに認識結果に出力する文字列を指定する．値は"[" および "]" で囲まれていること．省略した場合，第 1 フィールドの値がそのまま出力される．

また，単語 N-gram において，このフィールドを "{" "}" 囲みで記述した場合，その単語は「透過語」として扱われる．透過語については以下の節を参照のこと．

7.1.1.4 以降：音素列

以降は，その単語の読みを音素列として記述する．音素列は，空白で区切って記述する．

Julius の辞書形式は，単語の複数読みに対応していない．ある単語が複数の読みを持つ場合は，それぞれを異なる単語として別個に登録する．

以下に辞書エントリの例を示す．

```
課税+1  [カゼイ]        k a z e i
課題+1  [カダイ]        k a d a i
課長+1  [カチョウ]      k a ch o:
課長+1  [カチョウ]      k a ch o u
過ぎ+過ぎる+102 [スギ]  s u g i
過ぎ+過ぎる+114 [スギ]  s u g i
```

7.1.2 透過単語の指定について

試験的機能の一つとして，N-gram 用の辞書において，辞書上の任意の単語を「透過語」に指定することができる．透過語は，N-gram 確率計算時に，コンテキスト上でスキップされる．例えば，文「今日 は ， 良い」の前向き 3-gram 計算時，透過語指定がない場合は左のように通常の 3-gram が計算される．これに対して，単語「，」を透過語に指定した場合，右側の下 2 行のように,「，」が後続の単語のコンテキストから除外されて確率が計算される．このように，コンテキストとして情報が少ない単語を透過語にすることで，実質的な N-gram の距離を稼ぐことができる場合がある．

```
P(<s> 今日 は  ，寒い </s>)  P(<s> 今日 は  ，寒い </s>)
 =  P(<s> <s> | 今日)          =  P(<s> <s> | 今日)
  * P(<s> 今日 | は)              * P(<s> 今日 | は)
  * P(今日 は | ，)                 * P(今日 は | ，)
  * P(は ，  | 寒い)               * P(今日 は | 寒い)
  * P(，  寒い | </s>)            * P(は 寒い | </s>)
```

透過語は，主に句読点やフィラーなどコンテキストと無関係に挿入されやすい単語に対して指定すると効果がある場合がある．透過単語とするには，辞書において第 3 フィールドを "{" "}" 囲みで記述する．以下は単語「、+、+75」を透過語と指定する例である．

```
</s>  []  silE
<s> []  silB
、+、+75  {、}  sp
。+。+74  [。]  sp
？+？+73  [？]  sp
```

7.1.3 無音用単語の追加について

単語間のポーズに対応する単語を辞書に追加できる．単語間のポーズを言語モデル上でモデル化しておらず，辞書にもそれに対応するエントリが存在しない場合，このオプションを指定することで認識率が改善されることがある．追加する場合は`-iwspword` を指定する．また，追加される単語の内容はオプション `-iwspentry` で変更できる．

7.1.4 制約

デフォルトの語彙数の上限は 65,534 語である．ただし，コンパイル時に`--enable-words-int` を指定することで，単語 ID を 32bit に拡張してこの上限を撤廃できる（上限は理論上 2^31 語となる）．

1 単語あたりの音素数の最大は 256 である．

7.2 単語 N-gram

統計言語モデルのひとつである単語 N-gram を用いることができる．N の長さは 4.1.2 以前は 10-gram まで，4.1.3 以降は任意長の N をサポートする．

7.2.1 指定方法

使用するには，ARPA 標準形式の N-gram をオプション `-nlr`, `-nrl` で指定するか，あるいはバイナリ N-gram を`-d` で指定する．また単語辞書を `-v` で指定する．以下はバイナリ N-gram の場合の例である．

```
% julius ... -d bingramfile -v dictfile
```

7.2.2 前向き N-gram および後ろ向き N-gram

Julius は第 1 パスは left-to-right, 第 2 パスでは逆に入力終端から始端に向かって right-to-left に探索を行う．このため，第 1 パスでは通常の前向き (left-to-right) の 2-gram，第 2 パスでは後ろ向きの N-gram がそれぞれ必要となる．前向き，後ろ向きのどちらか一方のみを与える，あるいはそれぞれのパス用に両方の N-gram を指定することもできる．

前向きと後ろ向きの両方の N-gram を与えた場合，第 1 パスで前向きの 2-gram が適用され，第 2 パスで後ろ向きの N-gram が適用される．与えられた前向き N-gram に 3-gram より長い N-gram がある場合，その中の 2-gram のみが用いられる．内部でインデックスを共有するため，前向き N-gram と後ろ向き N-ram は同一の学習コーパスから学習され，語彙およびカットオフ値も同一である必要がある．

どちらか一方の N-gram のみを指定した場合，与えられたモデルと逆向きの探索パスでは，ベイズ則にしたがい算出した逆向き確率が用いられる．前向き N-gram のみの場合，第 1 パスではその中の前向き 2-gram を用い，第 2 パスではベイズ則にしたがって後向き確率に変換した確率が用いられる．逆に後ろ向き N-gram のみを指定した場合は，第 1 パスではその中の後ろ向き 2-gram をベイズ則により前向き 2-gram 確率に変換した値を用い，第 2 パスでは後ろ向き N-gram をそのまま適用する．

7.2.3 ファイル形式

N-gram を用いた認識では，N-gram ファイル，および各単語の読み（クラス N-gram では加えてクラス内確率）を記述した単語辞書の 2 つを Julius に与えて認識を行う．単語辞書では，第 1 フィールドに，対応する N-gram 中の語彙を指定する．以下，対応する N-gram のファイル形式を説明する．

7.2.3.1 ARPA 標準形式

ARPA 標準形式 (ARPA standard format) の N-gram 定義ファイルを読み込むことができる．前向き N-gram はオプション `-nlr`，後向き N-gram はオプション`-nrl` でそれぞれ与える．gzip で圧縮されたファイルも読み込める．

ARPA 標準形式の N-gram は，SRI-LM toolkit や CMU-Cam SLM Toolkit, palmkit などでテキストコーパスから学習することができる．SRILM で学習したモデルの場合は特別な注意が必要であるので，後述の「SRILM への対応」の節を参照のこと．

巨大な N-gram は読み込みに非常に時間がかかるので，次に説明する Julius バイナリ形式にあらかじめ変換しておくことが推奨される．

7.2.3.2 Julius バイナリ形式

Julius では独自のバイナリ形式をサポートしている．読み込み時間が ARPA 標準形式に比べて大幅に短縮できる．

ARPA 形式からの変換は，付属ツール mkbingram を用いる．前向き・後ろ向きの N-gram を同時に使用する場合は，ふたつをまとめて一つのバイナリ形式へ変換する．以下は使用例である．

```
% mkbingram -nlr forward_2gram.arpa -nrl backward_Ngram.arpa out.bingram
```

バイナリ形式の N-gram は，オプション -d で Julius に与えることができる．なお，gzip で圧縮されたファイルもそのまま読み込める．

7.2.4 SRILM への対応について

SRILM では，文開始記号について以下のような特殊な処理が行われるため，Julius で用いる際には特に注意と追加の手順が必要である．

- 文開始記号のユニグラムが常に "-99" となる．
- 文開始記号を出力とする N-gram が常に削除される．

Julius は逆向き探索を行うため，以上のような特徴を持つ SRILM 言語モデルをそのまま使用すると，逆向きに探索を行う際に言語制約上文頭記号が出現しないことになり，認識は起動するが，最後まで探索が成功せずに常に認識に失敗してしまう現象が発生する．

SRILM を用いて Julius 用の言語モデルを構築する手順を以下に示す．なお，SRILM で構築した言語モデルを Julius で使用する際には，前向きと後ろ向きの両方を用いる必要がある．

バージョン 4.1.2 以降では SRILM に対する対処が入っているため，必要な手順は 4.1.1 以前と 4.1.2 以降で異なる．以下，4.1.2 以降の場合の構築手順を示す．特にステップ 2 に注意せよ．

1. 学習コーパスを一行一文で用意する．
2. 各文の先頭・末尾に文開始記号"<s>" あるいは文終了記号"</s>" があれば，**それらを取り除く**．
3. SRILM で前向き 2-gram を学習する．オプションは必要に応じて適宜指定すること．

```
% ngram-count -order 2 ... -text train.text -unk -lm 2gram.arpa
```

4. 元コーパスから逆順コーパスを用意する．逆順コーパスとは文中の単語を逆順に並べたものであり，例えば以下のような perl スクリプトで元コーパスから生成できる．

```
#!/usr/bin/perl
while(<>) {
    print join(' ', reverse(split(/[ \t\n]+/))) . "\n";
}
```

5. 逆順コーパスから後ろ向き N-gram を学習する．コマンドは通常の前向きの作成時と同様に指定する．オプションは必要に応じて適宜指定すること．

```
% ngram-count -order 4 ... -text train-rev.text -unk -lm 4gram-rev.arpa
```

6. 学習した二つの N-gram を **mkbingram** で結合してバイナリ形式に変換する．

```
% mkbingram -nlr 2gram.arpa -nrl 4gram-rev.arpa 4gram.bingram
```

4.1.1 以前の Julius で用いる場合は，上記のステップ 5 までを行った後，以下の修正を人手で加えてからステップ 6 を実行する．

- 逆向き N-gram (上記例では 4gram-rev.arpa) を編集し，全ての N-gram 中の文開始記号"<s>" と文終了記号"</s>" を全て入れ替える．
- 両 N-gram 内で "<s>" と"</s>" の 1-gram 確率を調べ， "-99" となっているのがあれば，もう一方の出現確率に書き換える．

7.2.5 制約

- デフォルトの語彙数の上限は 65,534 語である．ただし，コンパイル時に`--enable-words-int` を指定することで，単語 ID を 32bit に拡張できる．この場合の上限は理論上 2^31 語となるが，メモリを多く消費する．
- Tuple 数の上限は，N-gram ごとに (2^32 - 1) 個である．
- バイナリファイルのファイルサイズの上限は 32bit OS では 4GB である．64bit OS ではこの制限は無い．
- 辞書上の単語に単語 N-gram 中に無い未知語が存在する場合，Julius は未知語エントリ（デフォルトは `<unk>` または`<UNK>`）の確率を未知語数で按分して割り当てる．辞書に未知語が存在するが，対応する未知語エントリが単語 N-gram 内に無い場合，Julius はエラー終了する．未知語エントリ名は Julius のオプション `-mapunk` で変更できる．

7.3 記述文法

Julius は，認識用の記述文法に基づく認識を行うことができる．認識用文法は，発話されうる文のパターン（構文および語彙）を形式言語の形で与えるものである．認識時には，与えられた文法上で可能な文パターンのなかから，入力に最もマッチする文候補が選ばれ，認識結果として出力される．文のパターンを人手で記述・作成するため，数十語〜数百語の小規模なタスクや，数字認識や住所認識など発話の制約が高い（ルール化しやすい）タスクに向いている．

文法に基づく認識では，言語モデルとして文法（構文制約および語彙辞書）に基づく認識を行う．文法は，BNF 形式に準じた独自のフォーマットで記述し，付属のコンパイラ mkdfa.pl(1) でオートマトン制約と辞書に変換してから Julius に与える．以下に詳細を述べる．

7.3.1 フォーマット

認識用文法は，以下の 2 種類のファイルとして記述する．

- grammar ファイル：構文制約を単語のカテゴリを終端規則として記述する．
- voca ファイル：カテゴリごとに単語の表記と読み（音素列）を登録する．

それぞれの書き方を以下に説明する．なお，例としてここでは,『りんご 3 個です』『蜜柑 8 個をください』などの発話を受理する「果物注文システム」用の文法を考える．また，変換後のオートマトン（dfa ファイル）についても述べる．

7.3.1.1 grammar ファイル

grammar ファイルでは，単語のカテゴリ間の構文制約（単語間の接続に関する制約）を BNF 風に記述する．`"S"` を開始記号として，1 行につき 1 つの書き換え規則を記述する．具体的には，左辺に書き換え元のシンボルを表記し， セミコロンで区切って右辺に書き換え後のシンボル列を列挙する．

この grammar ファイルにおいて，終端記号，すなわち左辺に現れなかったシンボルが，voca ファイルにおける「単語カテゴリ」となる．各単語カテゴリに属する実際の単語は，voca ファイルで記述する．

例として，前述の「果物注文システム」のための grammar ファイル `fruit.grammar` を以下に示す．

```
S       :  NS_B FRUIT_N PLEASE NS_E
# 果物名(+数量)
FRUIT_N :  FRUIT
FRUIT_N :  FRUIT NUM KO
# ください/にしてください/です
PLEASE  :  WO KUDASAI
PLEASE  :  NISHITE KUDASAI
PLEASE  :  DESU
```

この例では，`FRUIT`，`NUM`，`KO`，`WO`，`KUDASAI`，`NISHITE`，`DESU`，が終端記号，すなわち単語カテゴリとなり，これらに属する単語を voca ファイルで定義することになる．

以下は grammar ファイルの記述に関する注意点である．

- "#" で始まる行はコメント行で，任意のコメントを書くことができる．

- 英数字とアンダースコアが使用できる．大文字・小文字は区別される．
- "NS_B" と "NS_E" はそれぞれ文頭および文末の「無音区間」に対応する単語カテゴリである．文の最初と最後に必ず挿入すること．

なお，形式上は CFG のクラスまで記述可能だが，Julius はオートマトン（正規文法）のクラスまでしか扱えない．この制限はコンパイル時に自動チェックされ，オートマトンのクラスで扱えない場合はエラーとなる．

また，再帰性は左再帰性のみ扱える．1 回以上の繰り返しを許す文法を書く場合，以下のように記述する．

```
WORD_LOOP: WORD_LOOP WORD
WORD_LOOP: WORD
```

7.3.1.2 voca ファイル

voca ファイルでは，grammar ファイルで記述した終端記号（＝単語カテゴリ）ごとに単語を登録する．1 行に 1 単語ずつ，その単語の表記と発音音素列を記述する．以下に，前述の `fruit.grammar` に対応する voca ファイル `fruit.voca` を示す．

```
% FRUIT
蜜柑            m i k a N
リンゴ          r i N g o
ぶどう          b u d o:
% NUM
0          z e r o
1          i ch i
2          n i:
3          s a N
4          y o N
5          g o:
6          r o k u
7          n a n a
7          sh i ch i
8          h a ch i
9          ky u:
% KO
個              k o
% WO
を              o
% KUDASAI
ください        k u d a s a i
% NISHITE
にして          n i sh i t e
% DESU
です            d e s u
% NS_B
<s>             silB
% NS_E
</s>            silE
```

行頭が "%" で始まる行はカテゴリの定義である．定義するカテゴリは対応する grammar ファイルと一対一の対応がとれている必要がある．NS_B，NS_E には，それぞれ文頭・文末の無音に対応する無音音響モデル（Julius の標準音響モデルでは "silB", "silE"）を割り当てる．

7.3.2 コンパイル

grammar ファイルと voca ファイルを，付属のコンパイラ **mkdfa.pl** を用いて Julius の形式であるオートマトン（dfa ファイル）と単語辞書（dict ファイル）に変換する．`fruit.grammar`, `fruit.voca` を **mkdfa.pl** で `fruit.dfa` と `fruit.dict` に変換する実行例を以下に示す．

```
% mkdfa.pl fruit
fruit.grammar has 6 rules
fruit.voca    has 9 categories and 20 words
---
Now parsing grammar file
```

```
Now modifying grammar to minimize states[0]
Now parsing vocabulary file
Now making nondeterministic finite automaton[8/8]
Now making deterministic finite automaton[8/8]
Now making triplet list[8/8]
---
-rw-r--r--    1 foo      users         112 May  9 21:31 fruit.dfa
-rw-r--r--    1 foo      users         351 May  9 21:31 fruit.dict
-rw-r--r--    1 foo      users          65 May  9 21:31 fruit.term
```

7.3.3 コンパイルおよびチェックの方法

作成した文法がどの程度正しく作れているかをチェックする方法の一つとして，文法にしたがって文をランダムに生成するツール"generate" が提供されている．これを用いることで，意図に反するような文章が生成されないか，あるいは認識したい文章がきちんと出力されるかどうかチェックできる．以下は，"fruit.dfa", "fruit.dict" からの生成例である．

```
% generate fruit
Reading in dictionary...13 words...done
Reading in DFA grammar...done
Mapping dict item <-> DFA terminal (category)...done
Reading in term file (optional)...done
9 categories, 13 words
DFA has 8 nodes and 10 arcs
-----
 <s> 蜜柑 1 個 です </s>
 <s> リンゴ 9 個 です </s>
 <s> ぶどう 9 個 です </s>
 <s> リンゴ です </s>
 <s> ぶどう 4 個 です </s>
 <s> ぶどう にして ください </s>
 <s> ぶどう です </s>
 <s> 蜜柑 9 個 です </s>
 <s> 蜜柑 です </s>
 <s> 蜜柑 を ください </s>
```

7.3.4 Julius への指定方法

Julius に文法ファイル (.dfa, .dict) を与えるには，以下のようにオプション -gram を使用する．

```
% julius ... -gram basename
```

basename には，.dfa および .dict ファイルの共通のプレフィックスを指定する．例えば，"foo.dfa" と "foo.dict" がカレントディレクトリにある場合，"foo" のように指定する．また，以下のように別個に指定する旧バージョンでの方法も可能である．

```
% julius ... -dfa foo.dfa -v foo.dict
```

7.3.5 複数の文法を使用するには

Julius では，一つの言語モデルインスタンス内に複数の文法を使用できる．複数の文法が与えられたとき，Julius はそれらを並列に結合した文法を内部で生成し，それに基づいて認識を行う．

複数の文法を指定するには，オプション-gram においてコンマ区切りで複数の文法を指定する．

```
% julius ... -gram base1,base2,base3
```

あるいは，-gram を複数回に分けて指定することもできる．複数回指定した場合は，その全てが読み込まれる．

```
% julius ... -gram base1 -gram base2 -gram base2
```

また，文法のリストをファイルに記述しておき，それを与えることもできる．

```
% julius ... -gramlist gramlistfile
```

gramlistfile 内では一行に 1 つずつ文法のプレフィックスを指定する．プレフィックスは，その *gramlistfile* のある場所からの相対パスで記述する（実行時のカレントディレクトリではないことに注意）．また '#' 以降行末まではコメントとして無視される．この -gramlist についても，複数回指定したり，-gram と同時に指定した場合，その全てが読み込まれる．

-nogram オプションを使用すると，それ以前に-gram や -gramlist, -dfa, -dict で指定されていた文法をクリアすることができる．

複数の文法を指定した認識時，Julius は与えられた文法全体のなかで最も尤度の高い結果を出力する．オプション -multigramout を指定することで，文法ごとの最尤仮説を個別に出力することもできる．ただし，このオプションを指定した場合，第 1 パスのビーム幅が狭いと認識結果が得られない文法が出てくる．すべての文法に対して認識結果を得たい場合は，第 1 パスのビーム幅を十分広く取る必要がある．

7.3.6 文法における文中の短時間無音の指定

ある程度の長さの文章を発声しようとするとき，多くの場合，文節区切りなど特定の位置で，息継ぎによる「発声休止」が起こる．認識用文法において，この発声休止が起こる可能性のある場所を指定しておくことで，Julius は発声休止の有無を考慮した認識を行うことができる．

Julius で発声の休止位置を記述するには，まず grammar ファイルで，休止の出現しうる単語間にカテゴリ名「NOISE」を挿入する．さらに，voca ファイルでこの「NOISE」カテゴリに，無音の音響モデル（標準モデルでは "sp"）を読みとする単語を定義する．

以下に fruit での例を示す．休止が入りうる場所に入れていくのがよい．

```
# fruit.grammar (NOISE対応版)
S       :  NS_B FRUIT_N NOISE PLEASE NS_E
# 果物名(+数量)
FRUIT_N :  FRUIT
FRUIT_N :  FRUIT NUM KO
# ください/にしてください/です
PLEASE  :  WO NOISE KUDASAI
PLEASE  :  NISHITE KUDASAI
PLEASE  :  DESU
```

これに対応する fruit.voca では，以下の 2 行を追加する．第 1 欄は無音が挿入されたときに出力される文字列であり，何を指定してもよい．

```
% NOISE
<sp>  sp
```

なお，無音を表す無音音響モデルの名前のデフォルトは sp である．使用する音響モデルにこの名前のモデルが無音のモデルとして定義されている必要がある．モデル名が異なる場合，この名前は，オプション -spmodel で変更できる．

7.3.7 DFA ファイルの仕様

grammar ファイルと voca ファイルをコンパイルして得られる DFA ファイルはオートマトン定義ファイルであり，grammar で記述される文法制約を有限状態オートマトンに変換したものである．以下に上記の fruit 文法で生成されるファイル fruit.dfa を示す．

```
0 8 1 0 0
1 4 2 0 0
1 6 3 0 0
2 3 3 0 0
2 5 3 0 0
3 0 4 0 0
3 2 5 0 0
4 7 6 0 0
5 1 7 0 0
6 -1 -1 1 0
7 0 4 0 0
```

DFA は辞書上の単語カテゴリ番号を出力とする Mealy 型オートマトンであり，1 行に 1 つの遷移が書かれている．第 1 フィールドは遷移元の状態番号，第 2 フィールドは出力する単語カテゴリ番号，第 3 フィール

ドは遷移先の状態番号である．第 4 フィールドは第 1 フィールドの状態のフラグであり，最下位ビットが 1 であるとき最終状態を表わす．遷移を定義せずに状態フラグのみをセットしたい場合は，上記の最後から 2 行目のように，第 2，第 3 フィールドに -1 を指定する．第 5 フィールドはダミーであり，歴史的経緯から残されているが無視してかまわない．また，初期状態は常に状態番号 0 である．

単語カテゴリ番号と実際のカテゴリ名の対応は，コンパイル時に同時に出力される .term ファイルに記述されている．

7.4 単語リスト（孤立単語認識）

Julius は，音声コマンドや音声リモコンのような，より単純で簡単な音声認識を手軽に実現できるよう，単語リストのみで実行できる孤立単語認識を行える．

孤立単語認識を行う場合は，単語リストとして単語辞書ファイルをオプション `-w` で指定する．なお，文法と同様に複数の辞書を用いたり，個々の辞書を動的に有効化・無効化することができる．複数の辞書を指定するには，上記のオプションを繰り返し指定するか，あるいは辞書ファイルのリストが入ったファイルを `-wlist` で指定する．

```
% julius ... -w dictfile
```

Julius は，孤立単語認識のとき，入力発声の始終端の無音部分に対応するよう，無音モデルを各単語の前後に付加して認識を行う．具体的には，各単語の先頭に `silB`, 末尾に `silE` が自動的につけられる．なお，この値は以下のようにオプション `-wsil` で変更できる．

```
% julius ... -wsil head_sil_model_name tail_sil_model_name sil_context_name
```

`head_sil_model_name`，*`tail_sil_model_name`* はそれぞれ単語の先頭・末尾に付加される音響モデルの名前である．また，*`sil_context_name`* に NULL 以外を指定することで，トライフォン使用時のこれらの無音モデルのコンテキスト上の名前を指定できる．例えば，`-wsil silB silE sp` と指定した場合，`i m a` という単語は `silB sp-i+m i-m+a m-a+sp silE` として照合される．

7.5 ユーザ定義関数による言語制約拡張

Julius はアプリ上で定義した単語出現確率の計算関数を与えることで，ユーザが定義した任意の言語制約を使って認識を行うことができる．この場合，まずアプリ内であらかじめ指定された型の関数を定義し，それらを JuliusLib に対して，モデルの初期化前に引きわたしておく．そして，実行時には単語辞書を `-v` で指定し，さらにオプション `-userlm` を指定する．なお，同時に `-d` 等で単語 N-gram を与えることで，ユーザの計算関数への引数として単語 N-gram の確率を提供することも可能である．

このユーザ定義関数は，Julius 本体内で定義してもよいし，プラグインとして提供することもできる．

Chapter 8

認識アルゴリズムとパラメータ

Julius は，与えられた入力音声（特徴量系列）に対して，音響モデルと言語モデルのもとで確率が最大となる単語列を見つけ出す．本章では Julius がいかに音声認識処理を実行するか，その認識アルゴリズムについて概要を述べる．また，パフォーマンスチューニングのための種々の探索用パラメータ設定について解説する．

8.1 認識アルゴリズムの概要

Julius の音声認識アルゴリズムは，ツリートレリス探索方式を基礎とするアルゴリズムである．全体は 2 パス構成となっており，2 段階に分けて認識処理を行う．まず，第 1 パスでは入力全体に対して荒い認識処理を行い，有望な候補の集合をある程度絞り込む．このとき，簡便なモデルや近似計算を用いることで高速に処理を行う．第 2 パスでは詳細な認識処理を行うが，その際に第 1 パスの結果を参照しながら探索を行うことで，必要な部分にだけ精密な再計算を行って，最終的な最尤解を求める．このように複数回の照合によって段階的に候補を絞り込むことで，大語彙においても精度を落とさずに効率の良い認識を行うことを目標としている．

アルゴリズムの詳細を述べる．まず第 1 パスでは，音声入力と並行して，入力の始端から終端に向かって left-to-right にフレーム同期ビーム探索を行う．探索アルゴリズムは木構造化辞書に基づくものである．Julius では第 1 パスは第 2 パスのための前段処理であり，通常の 1 パス音声認識よりも強い近似を用いて高速性をある程度優先している．具体的には，単語履歴の 1-best 近似，N-gram における 1-gram factoring, 部分線形化辞書，単語間トライフォン近似を用いている．

第 1 パスで用いられる言語制約は言語モデルごとに異なる．単語 N-gram の場合は，その N-gram 中の前向き 2-gram が適用される．記述文法の場合は，文法から単語間の接続制約のみを抽出したカテゴリ対制約が用いられる．なお，このカテゴリ対制約は文法読み込み時に Julius 内部で自動生成される．

第 1 パスの結果として「単語トレリス」と呼ばれる単語候補集合が算出される．これは，第 1 パスの認識処理中の各フレームにおいて，ビーム内に終端状態が残った単語候補をインデックス化して保存したものである．それぞれの単語候補は，その始終端フレームと入力始端から単語末尾までの累積スコアを保持している．第 2 パスではこの単語トレリスを参照しながらもう一度探索が行われる．

第 2 パスは，第 1 パスとは逆に，入力の終端から始端に向かって後ろ向き（right-to-left）に探索を行う．アルゴリズムはいわゆるスタックデコーディングであり，仮説を保持するスタックにおいて最もスコアの高い仮説から順次単語単位で仮説を展開する．探索中の部分文仮説の評価スコアは，その時点までの（後ろ向きの）スコアに，未探索部分のヒューリスティックとして第 1 パスで生成された単語トレリスを接続した値となる．このスコアに従って best-first に探索を行い，最終的な認識結果を出力する．第 2 パスでは，第 1 パスの結果単語トレリス上に残った単語についてのみ計算を行うため，探索空間はあらかじめ絞り込まれており，詳細な計算を行うことができる．言語制約は全制約を用い，単語間や履歴近似も行わない．

一般に，パス間の中間表現としては上位 N 個の文候補や単語グラフがよく用いられる．これに対して，Julius の単語トレリスは候補を上位のみに絞り込まず，下位を含めて登場した候補全てを残す．このため，第 1 パスで強い近似によって計算誤差が増大しても，中間結果の中に最尤仮説系列が含まれ，そのため第 2 パスで誤差を回復できる可能性が高い．この単語トレリスの性質により，Julius の第 1 パスでは，Viterbi 経路の単語履歴依存性を考慮しない 1-best 近似や，1-gram factoring, 単語間の接続部における音素環境依存性の近似など，高速化のための様々な強い近似が用いられている．

なお，孤立単語認識の場合，単語間や単語履歴，言語モデルにかかる近似は必要なく，第 1 パスのみで正確な認識結果を出すことができる．このため，孤立単語認識では第 2 パスは実行されず，第 1 パスの結果が最終結果となる．

また，音声認識の計算の大部分は音響尤度計算が占めるが，Julius ではこの音響尤度計算について，混合ガウス分布の尤度を上位コンポーネントのみ求める Gaussian pruning を用いており，さらに尤度下位のトライフォンの尤度をモノフォンの尤度で代用する Gaussian mixture selection も利用可能である．

8.2 探索アルゴリズムにおける調節可能なパラメータ

認識アルゴリズムで調整可能なパラメータを以下に示す．これらは認識処理オプション（"-SR"）であり，モデルごとではなく認識処理インスタンスごとに設定する．

これらの探索パラメータは，認識精度を数%左右することもある重要なパラメータである．Julius のデフォルト値は，一緒に配布されている日本語標準モデル用の値であり，あくまで参考値である．与えられた言語モデル・音響モデルの組み合わせでの最大性能を求めるにはチューニングが必須である．たとえばモデルおよびその組み合わせによって最適な重みは異なるし，たとえば，計算時間よりも精度を重視する場合はビーム幅を広く取る必要がある．性能評価実験等を行う場合は特に注意されたい．

8.2.1 言語重みおよび挿入ペナルティ

統計言語モデルを用いた音声認識では，通常，言語モデルが与える尤度より，音響モデルの尤度のほうがダイナミックレンジが対数尤度で数倍程度大きく，扱う仮説が音響モデルに大きく引っ張られる．これを防いで言語制約を強調する目的で，言語モデルの尤度に一定の係数を乗じる言語重みが用いられる．

また，単語間の遷移に一定のペナルティを課すこともしばしば行われる．これは単語挿入ペナルティと呼ばれる．単語挿入ペナルティを大きく課すことで，単語間の遷移を抑制することができる．これは，例えば「派手に」という発話が助詞の「は で に」と認識されるといったように，短い単語仮説が連続して生成されてしまい湧き出し誤りを起こす場合に，このペナルティを設定することで長い単語を優先させるといった効果が期待できる．

N-gram では，`-lmp` *`weightpenalty`* で第 1 パスの，`-lmp2` *`weightpenalty`* で第 2 パスの（最終の）言語スコア重みと挿入ペナルティを指定できる．ペナルティは負であれば単語挿入を抑制し，正であれば単語挿入を促進する．以下は重みをそれぞれ 8.0, 挿入ペナルティをそれぞれ -1.0, 0.0 とする場合の例である．

```
% julius ... -lmp1 8.0 -1.0 -lmp2 8.0 0.0
```

文法では，`-penalty1` *`penalty`*, `-penalty2` *`penalty`* でそれぞれ第 1 パス，第 2 パスの単語挿入ペナルティを指定できる．

```
% julius ... -penalty1 -10.0 -penalty2 -10.0
```

言語重みと挿入ペナルティの設定は，認識精度に直接的に影響を与える．Julius では，使用するモデルの種類によって適当なデフォルト値が用いられるが，最適な認識精度を得るには，モデルの組み合わせごとに適切な重みを与える必要がある．

8.2.2 ビーム幅

第 1 パス，第 2 パスとも，解探索を行う際の仮説の足切り幅，すなわちビーム幅を設定できる．

ビーム幅の設定は，Julius の認識精度および処理時間に最も大きく影響する．小さいビーム幅を設定すると，計算する仮説の範囲が狭くなり，その結果処理は高速化される．ただし，小さくしすぎると最終的に一位になる解のパスが途中で足切りされる危険性が高まり，探索誤りによる認識エラーを生じる恐れが大きくなる．逆に，大きいビーム幅を指定すると，広い範囲を探索するため探索エラーが生じる可能性は低くなるが，そのぶん処理時間は長くなる．

第 1 パスのビーム幅は`-b` で指定できる．単位は木構造化辞書上のノード数（HMM 状態数）である．0 を指定すると全探索（ビーム幅＝木構造化辞書の全ノード数）となり，足切りを行わずに全てのノードを計算するようにすることもできる．

第 2 パスのビーム幅は，スタックデコーディングにおける単語単位の展開制限幅であり，`-b2` で指定できる．ある長さの単語展開回数がこの指定した上限値に達したら，それより単語数が短い仮説の展開を行わないようになる．これによって，スタックデコーディングが同じ場所で似た単語を展開しつづけて前に進まない状況を回避する．小さすぎる値は最尤仮説の発見を妨げ精度の低下を招くが，大きすぎると探索が終わりにくくなり，処理時間がかかるようになる．

8.2.3 第 1 パス

`-b` は第 1 パスのビーム幅を指定する．単位は木構造化辞書上のノード数（HMM 状態数）である．0 を指定すると全探索（ビーム幅＝木構造化辞書の全ノード数）となり，足切りを行わずに全てのノードを計算する．

8.2.4 第 2 パス

-b2 で第 2 パスのビーム幅を指定する．これはスタックデコーディングにおける単語単位の展開制限幅であり，ある長さの単語展開回数がこの指定した上限値に達したら，それより単語数が短い仮説の展開を行わないようになる．小さすぎる値は最尤仮説の発見を妨げ精度の低下を招くが，大きすぎると探索が終わりにくくなり，処理時間がかかるようになる．

-sb で第 2 パスの仮説尤度計算時のスコア幅を指定できる．仮説展開時に仮説ごとの音響尤度を更新するが，その際にフレームごとのそれまでの仮説の最大値から一定幅より下のパスを計算から除外することで音響モデル照合のコストを下げる効果が有る．小さい値にするほど第 2 パスの音響尤度計算量が少なくなるが計算誤りにより仮説のスコアが得られなくなる可能性が高くなる．最適値は音響モデルの対数尤度のレンジによる．

-s で仮説のスタックサイズを指定する．スタックサイズは，スタックデコーダが保持する仮説スタックのサイズである．小さすぎると探索誤りが起こりやすくなるため，大きい値を与えるとよいが，大きすぎるとメモリを多く消費する．

-m で第 2 パスの探索打ち切りのための仮説展開回数のしきい値を指定する．Julius は，指定した数の文仮説が見付かる前に仮説展開回数がこのしきい値に達したとき，探索を強制終了する．探索終了時に一つも文仮説が得られていない場合は，探索失敗（解なし）となる．大きい値を与えるほど，あきらめずに探索を続けるようになるが，計算時間がかかる．

-lookuprange は，第 2 パスの単語展開時に単語トレリス内から次単語候補を抽出する際のフレーム範囲を指定する．値を大きくするほど，周辺の多くの単語仮説を次単語候補として仮説展開するが，広くしすぎると余分な仮説を展開するため探索が前に進みにくくなる．

-looktrellis は，文法を用いた認識において，単語仮説を単語トレリス内に残った単語に限定するオプションである．Julius は文法ベースの認識の場合，次に文法上接続しうる仮説をすべて展開する．このとき，大語彙の文法，あるいは 1 音素からなるような短い単語が多い文法の場合，似た場所で大量の仮説が生成されて探索が進みにくくなり，第 2 パスの速度が通常よりも極端に遅く，認識精度も低くなることがたまにある．この場合，-looktrellis を指定することで展開仮説が絞られ，探索が安定化・高速化することがある．ただし，副作用として短い単語が削除される誤りが増大することがある．

8.2.5 その他のオプション

その他の探索に関するオプションを以下に列挙する．

- -1pass: 第 1 パスのみを実行する．
- -no_ccd, -force_ccd: 単語間トライフォンの近似計算を行うかどうかを明示的に指定する．デフォルトは，音響モデルがトライフォンであれば自動的にオンとなる．
- -iwsp: 任意の単語間における短時間ポーズの挿入を許す認識を行う．このオプションを指定すると，Julius 内部で全単語間にスキップ可能なショートポーズモデルを挟み込んだ認識が行われる．このオプションはマルチパスモード時のみ有効である．

  挟み込まれるショートポーズモデルの名前はオプション-spmodel で指定できる．このモデルは，前後の音素コンテキストを考慮されず，また前後の音素のコンテキストとしてもスキップされる．
- -transp: 通常の単語挿入ペナルティに加え，透過語の挿入ペナルティを指定することができる．デフォルトは 0 である．

8.3 認識結果の出力

Julius は通常，認識結果として最尤の候補一つを出力する．この結果出力を設定によって変更することが可能である．以下に出力可能な形式と結果の見方を解説する．

8.3.1 N-best リスト

-n で，指定された数の文仮説数が見付かるまで探索を行う．また -output で，その見つかった仮説のうち，実際に出力する上位仮説数を指定する．

N-gram を用いた認識では，ヒューリスティックの非適格性から最初に見つかる仮説が最尤でないことがしばしばある．このため，-n の値を 3 から 5 程度にしておくことで，最尤解を正しく得られる可能性が高くなる．大きい値を与えるほど第 2 パスの探索時間が長くなる．また，最上位以外の候補を出力したい場合は -output を指定する．

以下に "-n 3 -output 3" としたときの出力例を示す．"sentence" で始まる行が第 2 パスの認識結果の出力文字列，"wseq1" は言語モデルのエントリ列（N-gram では N-gram エントリ名，文法ではカテゴリ番号）である．"phseq" は音素並びであり，"|" は単語区切りを表す．また，"cmscore" は認識結果の各単語の単語信頼度を表す．"score" はその仮説の尤度であり，音響モデルの出力確率と重みづけられた言語モデルの出力確率の対数尤度の和である．別々にスコアを出力したい場合は，オプション"-separatescore" を指定する．それぞれにおいて，後ろの数字が順位を表す．

```
sentence1: 個人技 が 、 随所 で 光っ た 。
wseq1: <s> 個人技+コジンギ+2 が+ガ+58 、+、+75 随所+ズイショ+2 で+デ+58 光っ+ヒカ ↩
    ッ+光る+44/17/8 た+タ+70/47/2 。+。+74 </s>
phseq1: silB | k o j i N g i | g a | sp | z u i sh o | d e | h i k a q | t a | sp ↩
     | silE
cmscore1: 0.441 0.312 0.286 0.162 0.233 0.406 0.339 0.419 0.813 1.000
score1: -5333.988281
sentence2: 個人技 が 随所 で 光っ た 。
wseq2: <s> 個人技+コジンギ+2 が+ガ+58 随所+ズイショ+2 で+デ+58 光っ+ヒカッ+光 ↩
    る+44/17/8 た+タ+70/47/2 。+。+74 </s>
phseq2: silB | k o j i N g i | g a | z u i sh o | d e | h i k a q | t a | sp | ↩
    silE
cmscore2: 0.441 0.327 0.648 0.233 0.406 0.339 0.419 0.813 1.000
score2: -5342.767578
sentence3: 個人技 が 、 随所 で 勝っ た 。
wseq3: <s> 個人技+コジンギ+2 が+ガ+58 、+、+75 随所+ズイショ+2 で+デ+58 勝っ+カ ↩
    ッ+勝つ+44/13/7 た+タ+70/47/2 。+。+74 </s>
phseq3: silB | k o j i N g i | g a | sp | z u i sh o | d e | k a q | t a | sp | ↩
    silE
cmscore3: 0.441 0.312 0.286 0.162 0.231 0.408 0.075 0.419 0.813 1.000
score3: -5343.241211
```

なお，第 2 パスの再探索は第 1 パスで残った仮説集合に対して行われるため，多くのバリエーションの仮説を残すには，第 1 パスのビーム幅も大きくしておくほうがよい．

8.3.2 単語ラティス形式

認識結果の上位仮説集合を，単語グラフ（ラティス）形式で出力できる．Julius の単語ラティス生成アルゴリズムは，第 2 パスのスタックデコーディング中に有望な単語仮説を逐次保存するものである．

オプション -lattice で単語ラティスの出力が行える．ただし，このオプション指定時，第 2 パスの探索アルゴリズムの一部が，グラフ出力を考慮したものに変更される．このため，同時に出てくる N-best 文候補は，グラフ出力を行わない通常の N-best とは異なることがある点に注意されたい．

十分な大きさのグラフを得るには，第 1 パスで多くの候補を残し，かつ第 2 パスでもある程度以上長く行う必要がある．-b, -b2 に大きい値を指定し，-n も小さすぎるとグラフが生成されないので 5 以上程度の値を指定する方がよい．

グラフの生成に関するパラメータがいくつか設定できる．オプション-graphrange は，同一単語をマージするオプションである．"-1" を指定すると仮説のマージを行わない（たとえ同一位置に同一単語があっても，左右のコンテキストが異なれば異なる仮説として扱う）．0 以上の値を指定すると，始終端フレームのずれがそれぞれ指定値以内である同一の単語について，尤度の高い方にマージする．値を大きくするほど小さいグラフが生成されるが，コンテキストの情報が失われる．また-graphcut はグラフの深さのカットオフ値である．

-graphboundloop はマージにおける dead loop 回避のための繰り返し上限数，-graphsearchdelay は単語候補生成タイミングに関するオプションである．この 2 つは通常は変更する必要はない．

出力例を以下に示す．[1] 1 単語ごとに以下のような情報が出力される．なお，実際には 1 単語あたり一行で出力されるが，ここでは説明のために適当に改行している．

```
24: [96..129]
   left=21 right=31,29
   left_lscore=-13.968800 right_lscore=-37.530403,-36.043201
   wid=11615 name="受賞" lname="受賞+ジュショー+17"
   f=-5320.787109 f_prev=-5327.085449
   g_head=-2937.868652 g_prev=-2078.347656
```

[1] なお，ラティス出力でもラティスとは別に上位 N 個の文候補が出力されるが，通常の N-best 時と探索アルゴリズムが異なるため，通常の N-best と同じではないので注意されたい．

```
   forward_score=-102.234932 backward_score=-120.460266
   AMavg=-25.326530
   cmscore=0.072092 graphcm=0.119256
   headphone=a-j+u tailphone=sh-o:+d
```

最初はこの単語の通し番号 (ID) であり，続く括弧内はマッチしたフレーム区間である．left, right は左右に接続する単語のリスト，left_lscore, right_lscore はそれぞれの言語スコアを表す．wid は辞書上の単語番号，name は出力文字列，lname は言語モデルのエントリである．

Julius は第 2 パスの探索中にグラフを生成する．f, f_prev, g_head, g_prev はその探索中にこの単語が得られた際の仮説スコアを表している．n をこの単語としたとき，それぞれ左端での仮説スコア (g(n) + h(n+1)), 右端での仮説スコア (g(n-1) + h(n))，左端での累積 Viterbi スコア g(n), 右端での累積 Viterbi スコア g(n-1) + LM(n) をそれぞれ表す．ただし g(n) は単語 n の第 2 パスでの right-to-left スコア，h(n) は第 1 パスでの left-to-right スコア，LM(n) は言語確率を表す．

julius はラティスが生成されたあとに再び forward / backward algorithm を適用して，各単語ごとのグラフ上の事後確率に基づく信頼度を算出する．forward_score, backward_score および graphcm がそれにあたる．また，cmscore は本来の Julius の計算で求められる信頼度である．

headphone, tailphone は左コンテキストと右コンテキストを考慮した左端音素と右端音素である．仮説がマージされたときは，最もスコアの高かった単語の情報が格納される．

8.3.3 Confusion network

オプション -confnet で，認識結果を confusion network の形で出力できるようになった．Julius の内部では，まず単語グラフを生成してから，仮説をマージしながら confusion network を生成する．

十分な大きさのグラフを得るには，単語グラフと同様，第 1 パスで多くの候補を残し，かつ第 2 パスでもある程度以上長く行う必要がある．-b, -b2 に大きい値を指定し，-n も小さすぎるとグラフが生成されないので 5 以上程度の値を指定する方がよい．

8.3.4 漸次出力

第 1 パスの計算中に，その時点での最尤パスを漸次的に出力することができる．-progout を指定すると，一定時間おきに，その時点での最尤候補を出力する．インターバルはオプション-proginterval でミリ秒単位で指定できる．

8.3.5 バージョン 3 との出力形式の互換性について

バージョン 4 で変更された重要な仕様の一つに，探索失敗時の結果出力がある．バージョン 3.x では，第 2 パスの探索が失敗したとき，第 1 パスの最尤仮説をそのまま最終的な認識結果として代用して出力していた．これに対して，バージョン 4 以降では，第 2 パスの探索失敗時には,「認識誤り」として結果が出力されないようになった．

バージョン 3.x と同じように第 1 パスの結果を代用出力したい場合は，オプション "-fallback1pass" を指定する．これにより，このオプションを指定することで，第 2 パスの失敗時に，第 1 パスの最尤仮説を最終結果として出力することができる．

8.4 アラインメント出力

Julius は認識結果の候補文を入力に対して再照合し，単語や音素ごとの対応区間を求めるアラインメント（forced alignment）が行える．結果として，単語や音素などの単位ごとに，マッチする区間の始終端フレームとその区間内の平均音響尤度が出力される．

単語単位のアラインメントは "-walign", 音素単位は"-palign", HMM の状態単位は"-salign" を指定する．複数を同時に選択することもできる．N-best 出力の場合，候補ごとにアラインメントが行われる．

以下は "-walign" の出力例である．最初のカッコ内がマッチした区間（フレーム単位），次が区間内の音響尤度のフレーム平均，言語エントリ文字列，出力文字列となる．

```
=== begin forced alignment ===
 id: from  to    n_score    unit
 -----------------------------------------
[   0   23]  -21.430922  <s>    []
[  24   65]  -24.608768  個人技+コジンギ+2      [個人技]
[  66   91]  -25.395161  が+ガ+58       [が]
[  92   97]  -29.494751  、+、+75       [、]
```

```
[  98  131]  -25.065617  随所+ズイショ+2          [随所]
[ 132  139]  -25.297760  で+デ+58          [で]
[ 140  170]  -25.976215  光っ+ヒカッ+光る+44/17/8          [光っ]
[ 171  185]  -26.640495  た+タ+70/47/2  [た]
[ 186  195]  -23.992529  。+。+74          [。]
[ 196  208]  -22.155687  </s>   []
re-computed AM score: -5172.585449
=== end forced alignment ===
```

8.5 単語信頼度

認識結果の各単語について，信頼度と呼ばれる指標を出力することができる．単語信頼度は事後確率に基づいて算出され，0 から 1 の値を取る．1 に近いほど，競合候補に比べて尤度の差が大きかったことを表し，認識結果として「信頼度が高い」と言える．また，0 に近い場合，似た確率を持つ多くの競合候補が存在したことを表し，認識結果として「信頼が低い」と見なせる．この信頼度を用いて，例えば低信頼度部分の除外や，高信頼度語のみから頑健な意図理解をする，といった応用が考えられる．ただし，環境や語彙の影響を受けやすく，特に大語彙環境では信頼性が低い傾向にある．

信頼度は認識結果の出力で "cmscore" で始まる行に，1 単語ずつ出力される．

8.6 認識処理に基づく入力区切り

Julius は，認識中の仮説情報に基づいて入力を区切って認識することができる．息継ぎなどの短時間無音で入力を細かく区切りながら逐次認識する「ショートポーズセグメンテーション」，さらにそれを無音区間をスキップするよう拡張した「デコーダベース VAD」を実装している．以下にそれぞれについて説明する．

なお，Julius では「無音単語」の尤度を基準に区間判定を行う．この無音単語とは，無音音響モデルのみを読みとする辞書上の単語である．無音音響モデルは名前で指定でき，デフォルトは "sp" である（オプション -spmodel で変更可能）．また，N-gram を用いた認識では文頭・文末の無音モデル（デフォルト："silB", "silE"，-silhead および-siltail で指定可能）も無音音響モデルとして扱われる．また，上記で指定できる以外の無音音響モデル名は-pausemodels で列挙指定できる．

8.6.1 ショートポーズセグメンテーション

オプション "-spsegment" を指定すると，ある 1 入力を認識する際に，短時間の無音区間が現れるたびに細かく区切って順次確定させていくことができる．Julius のこの機能は「ショートポーズセグメンテーション」と呼ばれる．これによって，長時間の音声入力を自動的に区切りながら逐次的に認識を可能にする．

ショートポーズセグメンテーション有効時，Julius は辞書中の無音単語の仮説尤度から短時間無音区間の判定を行う．Julius は第 1 パスにおいてフレームごとの最尤仮説をチェックし，無音単語が一位仮説となるフレームが一定時間以上続くとそれを無音区間と判定し，入力をそこで区切る．そして，区切られた区間について第 2 パスを実行し，仮説を確定した後，区切った部分から（のりしろ分少し遡って）認識を再開する．これによって，短い無音区間で認識処理を区切って確定しながら漸次的に認識を進めていくことができる．

8.6.2 デコーダベース VAD

ショートポーズセグメンテーションは，Julius を configure に--enable-decoder-vad をつけてコンパイルすることで，さらにデコーダベース VAD に拡張される．上記オプションをつけてコンパイルした Julius を -spsegment をつけて起動すると，Julius はデコーダベース VAD を行う．

デコーダベース VAD では，ショートポーズセグメンテーションと同じ方法で無音区間の検出を行うが，その検出した無音区間が長い場合，認識処理は行われるがその間仮説の生成は抑制される．無音単語以外の単語が一位仮説になった瞬間，仮説の生成が再開され，その場から次の区間認識が始まる．

関連オプションとして -spdur, -spmargin, -spdelay がある．詳しくはマニュアルを参照のこと．

Chapter 9

複数モデルを用いた認識

Julius は 4.0 より，複数のモデルの並列音声認識をサポートしている．複数の言語モデルや音響モデルを用いて，入力に対して並列に認識を行い，複数の結果を一度に得ることができる．

複数のモデルを用いるには，オプション指定時に，モデルやパラメータの指定の前にインスタンス宣言 -AM, -LM, -SR を行う．ここではインスタンス宣言の書式，および複数インスタンスの宣言方法について述べる．

9.1 インスタンスの宣言

9.1.1 音響モデルインスタンス (-AM)

-AM name

-AM は，*name* という名前の音響モデルインスタンスを新たに宣言する．-AM が指定されるたびに新たなインスタンスが生成され，それ以降にある音響モデルおよび特徴量抽出パラメータは，そのインスタンスに格納される．以下は２つの音響モデルを用いる場合のインスタンス宣言の例である．

```
-AM AM1
....(音響モデル1関連の設定)
-AM AM2
....(音響モデル2関連の設定)
```

インスタンス指定を一つも行わない場合，暗黙のインスタンス名は_default となる．ただし，Julius は -AM が指定されている場合，最初の -AM より前にあるパラメータは破棄してしまう．複数モデルを用いる場合は，必ず上記のように各モデルごとに最初に -AM で宣言する必要がある．

音響モデルインスタンスに格納されるパラメータは，音響モデルのファイルや計算アルゴリズムに関する設定，ならびにそのモデルで使用する特徴量の設定である．Julius は音響モデルごとに個別に特徴量抽出を行える．このため，複数の音響モデルを用いる場合，特徴量抽出のパラメータ設定は，モデルごとに行う必要がある点に注意されたい．たとえすべてのモデルで同一の特徴量を用いる場合でも，その特徴量抽出の設定をすべての音響モデルインスタンスで繰り返し記述する必要がある．なお，格納されるオプションの完全なリストは，マニュアルの「音響モデルおよび特徴量抽出」のセクションにある．

GMM については，オプション-AM_GMM を指定することで，それ以降に GMM のための特徴量抽出の設定を定義できる．-AM_GMM が指定されない場合，GMM は最初に定義された音響モデルインスタンスの特徴量を共有する．

-AM_GMM

なお，Julius は１つの音声入力をモデル間で共有するため，下記のオプションについては全インスタンスで同一の値が設定されている必要がある．Julius は，起動時にこれらのパラメータをチェックし，同一の値が指定されていない場合エラーとする．

- 入力音声のサンプリングレート (-smpFreq または -smpPeriod, あるいは HTK Config 内の SOURCERATE)
- フレームシフト (-fshift)
- フレームサイズ (-fsize)

9.1.2 言語モデルインスタンス (-LM)

-LM は言語モデルのインスタンス宣言である．いくつかの言語モデルタイプ（単語 N-gram, 文法，単単語認識等）が指定できる．一つのインスタンスには１つの言語モデルタイプのみ指定できる．

9.1.3 認識処理（デコーディング）インスタンス (-SR)

-SR は認識処理単位であり，上記のモデルインスタンスを参照しながら実際に認識を行う処理インスタンスを定義する．参照する音響モデルインスタンス名と言語モデルインスタンス名をそれぞれ指定し，続けて認識処理のためのパラメータ（重み・ビーム幅等）を指定する．

9.2 オプションの記述位置について

インスタンス宣言を用いた設定では，各指定オプションは，その時点で宣言されているインスタンスについての指定と解釈される．このため，各種オプションは，対応するインスタンス宣言の後に書く．例えば，音響モデルおよび特徴量関連のオプションは-AM の後に，言語モデル関係は-LM のあとに，デコーディング用パラメータは-SR のあとに書く必要がある．また，音声入力の設定などの全体オプションは，最初のインスタンス宣言の前か，あるいはオプション-GLOBAL の後に書く．以下に例を示す．

```
(global options)
-AM am1
(AM related options)
-LM lm1
(LM related options)
-SR search1 am1 lm1
(Search related options)
-GLOBAL
(global options)
```

-SR の後に LM オプションを記述するなど，異なる場所にオプションを書くとエラーとなる．どのオプションがどのインスタンスの種類に対応するかは，リファレンスマニュアルを参照のこと．

このオプション位置の制約および -GLOBAL オプションは，バージョン 4.1 で導入された．それ以前の Julius-4.0 では，上記の制約は存在せず，例えば -LM の後に AM オプションを記述するなど自由な記述ができた．しかし，オプションとインスタンスの関係があいまいになるため，上記のようなセクション付けが行われるようになった．4.0 での仕様に戻したい場合は，オプション -nosectioncheck を指定する．これにより，4.0 で動いている jconf をそのまま使うこともできる．

9.3 インスタンス宣言を用いた Jconf ファイルの例

以下は，１つの音響モデルと複数の言語モデルを用いる場合の例である．AM1, LM1, LM2, SR1, SR2 は任意のインスタンス名である．

```
-AM AM1
....(音響モデル関連の設定)
-LM LM1
....(言語モデル１および特徴量抽出関連の設定)
-LM LM2
....(言語モデル２および特徴量抽出関連の設定)
-SR SR1 AM1 LM1
....(AM1 + LM1 の認識処理の設定)
-SR SR2 AM1 LM2
....(AM1 + LM2 の認識処理の設定)
```

あるインスタンス内のパラメータは，他のインスタンスとは共有されずに完全に独立している．このため，特徴量抽出や言語モデル重みなど，モデルに付随するパラメータは全てインスタンスごとに改めて定義する必要がある点に注意が必要である．

複数モデルの指定を行う場合，全てのモデルにおいてまず最初に上記のように名前を宣言すること．最初のインスタンス宣言の前にあるパラメータは，グローバルなパラメータを除いて，無視される．

また，4.0 現在では，複数の音響モデルに対して一つの言語モデルを用いる組合せは直接サポートされていない．これは，言語モデルの一部の処理が音響モデルに依存しているためである．この制限は，同じ言語モデルを複数指定することで回避できる．例えば，以下のように設定を行いたい場合，

```
-AM am_1 -AM am_2
-LM lm (LM spec..)
-SR search1 am_1 lm
-SR search2 am_2 lm
```

以下のように同じ言語モデルを複数回指定してそれぞれの音響モデルに割り付ければよい.

```
-AM am_1 -AM am_2
-LM lm_1 (LM spec..)
-LM lm_2 (same LM spec..)
-SR search1 am_1 lm_1
-SR search2 am_2 lm_2
```

Chapter 10

モジュールモード

Julius をモジュールモードで起動することで，Julius を音声認識サーバーとして動かすことができる．モジュールモードで起動された Julius は，TCP/IP 経由でクライアントと接続し，クライアントへの認識結果や音声イベントの送信，およびクライアントからの動作制御を行うことができる．この章では，このモジュールモードについて解説する．

10.1 基本動作

Julius は `-module` を指定することでモジュールモードで起動する．モジュールモードでは，起動後，クライアントからの TCP/IP 接続待ちとなる．デフォルトのポート番号は 10500 であるが，`-module` *`portnum`* のようにポート番号を変更することができる．

クライアントからの接続を受けると，Julius は音声認識可能な状態となる．音声入力に対して音声認識を行いながら，クライアントへ認識結果を送信する．また，音声入力の開始・終了などのイベントを随時送出する．クライアントからは，認識の一時停止や再開といった動作制御，認識用文法の追加や削除などが行える．また，複数モデルによるマルチデコーディング時は，認識処理インスタンスごとの一時停止なども行える．

クライアントとの接続は一対一を想定しており，複数クライアントの同時接続には対応していない．クライアントとのネットワークソケットが切断されると，Julius は認識を停止して，次のクライアントが接続するまで待ち状態となる．

10.2 サンプルクライアント jcontrol による動作確認

Julius にはサンプルのクライアント jcontrol が付属している．これは Julius から送信されたメッセージをそのまま標準出力に出力し，またいくつかの簡単なコマンドを送ることができるツールである．これを使い，以下の要領で Julius のモジュール動作を試すことができる．

```
(1) サーバ：Julius を通常の起動方法に "-module" オプションを追加して起動

    % julius -C fast.jconf -module

(2) クライアント：jcontrol を以下のように起動

    % jcontrol (1)を実行しているホスト名
```

この状態で音声認識を行うと，結果が jcontrol 側に出力される．また，jcontrol で "pause" と入力して enter と押すと認識中断，"resume"で認識が再開できる．

以下，プロトコルについて詳細を述べる．なお，実際の実装については，jcontrol のソースコード，および `julius/module.c`, `julius/output_module.c` も大いに参考にされたい．

10.3 クライアントへの出力メッセージ仕様

Julius からクライアントへは，ソケット経由で XML 形式のテキストメッセージが送られる．文字エンコーディングは，認識に使用する言語モデルの内容がそのまま送られる．出力文字列のエンコーディングはオプ

ション-charconv で変えることができる．たとえば日本語 shift-jis の出力を UTF-8 に変えるには，起動時に -charconv sjis utf8 と指定する．

メッセージは XML 形式で構成される．改行コードは"\n"である．クライアントがパーズしやすくするため，１回のメッセージ送信ごとに，データの終端として，"." のみの行が送信される．以下はデータの例である．

```
<STARTPROC/>
<INPUT STATUS="LISTEN" TIME="994675053"/>
<INPUT STATUS="STARTREC" TIME="994675055"/>
<STARTRECOG/>
<INPUT STATUS="ENDREC" TIME="994675059"/>
<GMM RESULT="adult" CMSCORE="1.000000"/>
<ENDRECOG/>
<INPUTPARAM FRAMES="382" MSEC="3820"/>
<RECOGOUT>
  <SHYPO RANK="1" SCORE="-6888.637695" GRAM="0">
    <WHYPO WORD="silB" CLASSID="39" PHONE="silB" CM="1.000"/>
    <WHYPO WORD="上着" CLASSID="0" PHONE="u w a g i" CM="1.000"/>
    <WHYPO WORD="を" CLASSID="35" PHONE="o" CM="1.000"/>
    <WHYPO WORD="白" CLASSID="2" PHONE="sh i r o" CM="0.988"/>
    <WHYPO WORD="に" CLASSID="37" PHONE="n i" CM="1.000"/>
    <WHYPO WORD="して" CLASSID="27" PHONE="sh i t e" CM="1.000"/>
    <WHYPO WORD="下さい" CLASSID="28" PHONE="k u d a s a i" CM="1.000"/>
    <WHYPO WORD="silE" CLASSID="40" PHONE="silE" CM="1.000"/>
  </SHYPO>
</RECOGOUT>
.
```

<SHYPO> のところは認識結果の文単位の情報である．SCORE が音声認識エンジンの出した尤度，GRAM は文法使用時に，この文章が属する文法番号である．WORD, CLASSID, PHONE, CM はそれぞれ認識結果の単語表記，言語エントリ（N-gram の単語エントリあるいは文法のカテゴリ番号），音素列，単語確信度を表す．認識失敗時は <RECOGOUT> ... </RECOGOUT> の代わりに<RECOGFAIL/> が出力される．

Julius からクライアントへ送信されるメッセージの一覧を，表 10.1 に示す．また，認識結果出力の詳細を表 10.2 に示す．

10.4 クライアントから受信できる命令コマンド

クライアントから Julius にソケット経由でコマンドを送ることで，Julius を制御することができる．コマンドは，コマンドの文字列を末尾に改行コード"\n" をつけて送信する．そのコマンドが引数を必要とする場合は，そのコマンド文字列の次の行として，引数を送る．

ADDGRAM, CHANGEGRAM では，クライアントから Julius へ文法を送信する．クライアントは，コマンド文字列の行につづけて，文法ファイル (.dfa) と辞書ファイル (.dict) の中身を Julius へ送信する．各ファイルの最後には，"DFAEND" あるいは"DICEND" のみの行をそれぞれ送る．また，これらのコマンドのみ，送信する文法の名称をコマンド文字列の直後に空白につづけて指定する．ADDGRAM の送信内容の概要を以下に示す（CHANGEGRAM も同じである）．

```
ADDGRAM 新文法名
  ［文法ファイル (.dfa) の内容］
DFAEND
  ［辞書ファイル (.dict) の内容］
DICEND
```

Julius が複数モデルを使って認識処理インスタンスを同時に走らせるマルチデコーディングを行っている場合，プロセス関係および文法操作は「カレントインスタンス」について行われる．全処理インスタンスの一覧は LISTPROCESS で得ることができ，カレントインスタンスは，CURRENTPROCESS や SHIFTPROCESS で変更できる．また，ACTIVATEPROCESS や DEACTIVATEPROCESS を使うことで，個々の認識処理インスタンスを無効化・有効化できる．

コマンドの一覧を表 10.3 および表 10.4 に示す．

Table 10.1 モジュールモードの送信メッセージ

内容	意味	送信タイミング
<STARTPROC/>	認識エンジン動作開始	起動時，あるいは一時停止状態からの再開など，認識エンジン全体が動き始めたとき
<ENDPROC/>	認識エンジン停止時	一時停止時やエラー時など，認識エンジン全体が停止したとき
<STARTRECOG/>	認識処理開始	認識対象となる入力区間を検知し，第１パスの認識処理をスタートした時点
<ENDRECOG/>	認識処理終了	現在の区間の認識処理が完了した時点
<INPUT STATUS="LISTEN" TIME="..."/>	入力開始	音声入力部が，入力を開始できる状態になったとき
<INPUT STATUS="STARTREC" TIME="..."/>	入力始端検知	音声入力部が，音入力の開始を検知したとき
<INPUT STATUS="ENDREC" TIME="..."/>	入力終端検知	音声入力部が，音入力の終端を検知したとき
<INPUTPARAM FRAMES="..." MSEC="..."/>	入力長情報	現在認識中の区間の入力長が確定した時点
<GMM RESULT="xxx" CMSCORE="..."/>	GMM 最尤のモデルとその GMM 信頼度	入力終了後，GMM による入力全体の尤度が得られた時点
<RECOGOUT>...</RECOGOUT>	認識結果（成功時）	認識結果が得られたとき
<RECOGFAIL/>	認識失敗	最尤解が得られず解なしとなった場合．入力が極端に短かったり，入力内容が認識文法と大きくかけ離れている場合など．
<REJECTED REASON="...">	入力棄却	GMM や入力長制限などによって入力が棄却されたとき
<GRAPHOUT>...</GRAPHOUT>	単語グラフ	単語グラフ出力が指定されている場合，認識成功時
<GRAMINFO>...</GRAMINFO>	現在エンジンが保持している文法情報	文法切り替え時，および SYNCGRAM コマンド受信時
<SYSINFO PROCESS="ACTIVE|SLEEP">	エンジンの現在の状態	STATUS コマンド受信時
<ENGINEINFO TYPE="Julius" VERSION="4.1" CONF="fast"/>	エンジンのバージョン	VERSION コマンド受信時
<GRAMMAR STATUS="RECEIVED"/>	文法受け取り確認	クライアントから文法データを受け取ったとき
<GRAMMAR STATUS="READY"/>	文法準備完了確認	SYNCGRAM コマンドを処理終了し，文法の更新が終了したとき
<GRAMMAR STATUS="ERROR" REASON="..."/>	エラー	文法関連のコマンドでエラーが発生したとき
<RECOGPROCESS>...<RECOGPROCESS>	認識処理インスタンスの情報	プロセス関連のコマンド実行時

Table 10.2 モジュールモードの認識結果出力の詳細

出力	意味
<RECOGOUT>...</RECOGOUT>	認識結果全体
<SHYPO>...</SHYPO>	文候補．RANK は順位，SCORE は対数尤度（音響スコア＋言語スコア），GRAM は複数文法使用時にその候補の属する文法の ID をそれぞれ表す．
<WHYPO>...</WHYPO>	文候補内の単語候補．WORD は出力文字列，CLASSID は言語エントリ（N-gram では N-gram エントリ，文法ではカテゴリ番号），PHONE は音素列，CM は単語信頼度をそれぞれ表わす．

Table 10.3 クライアントから送信できるコマンド（共通）

コマンド	引数	Julius の動作
STATUS	-	エンジンが現在動作中であるかどうかを返す
DIE	-	Julius を強制終了する
VERSION	-	Julius のバージョンを返す
PAUSE	-	エンジンを一時停止する．受信時点で入力待ちだったときは即座に停止する．認識を実行中の場合，その入力が終わり認識が終了してから停止する．
TERMINATE	-	エンジンを一時停止する．入力待ち・認識実行中にかかわらず，即時停止する．
RESUME	-	一時停止状態から復帰してエンジン動作を再開する
CURRENTPROCESS	認識処理インスタンス名	カレントインスタンスを指定された名前のインスタンスに変更
SHIFTPROCESS	-	カレントインスタンスを次の認識処理インスタンスに変更
ADDPROCESS	jconf パス名（Julius から見えるパスで）	エンジンに新たに認識処理インスタンスを追加する
DELPROCESS	認識処理インスタンス名	指定されたインスタンスをエンジンから削除する
LISTPROCESS	-	すべての認識処理インスタンスの情報を出力する
DEACTIVATEPROCESS	認識処理インスタンス名	指定されたインスタンスを一時無効化する
ACTIVATEPROCESS	認識処理インスタンス名	指定されたインスタンスを有効化する

Table 10.4 クライアントから送信できるコマンド（カレントインスタンスが文法の場合）

コマンド	引数	Julius の動作
GRAMINFO	-	エンジンが現在持つ文法情報を返す
CHANGEGRAM 名前	文法データ（DFA+辞書）	文法データを受け取り，エンジンの現在保持する文法をそれに入れ替える．
ADDGRAM 名前	文法データ（DFA+辞書）	文法データを受け取り，エンジンに追加する．
DELGRAM	数字もしくは文法の名前（複数の場合空白で区切る）	指定された文法をエンジンから削除する．
DEACTIVATEGRAM	数字もしくは文法の名前（複数の場合空白で区切る）	指定された文法を一時的に無効化する
ACTIAVETGRAM	数字もしくは文法の名前（複数の場合空白で区切る）	指定された文法を有効化する
INPUTONCHANGE	"TERMINATE", "PAUSE" あるいは "WAIT"	文法をクライアントから受信したときに，指定されたタイミングで文法を更新する
SYNCGRAM	-	文法の更新を今すぐ行う
ADDWORD	文法 ID，その後辞書エントリ	受け取った辞書エントリを指定された文法に追加する

Chapter 11

プラグイン

Julius-4.1 以降では，プラグインにより Julius に様々な機能や処理を追加することができる．プラグインは，関数を定義する共有オブジェクト（あるいは DLL）ファイルであり，Julius 実行時に動的に組み込まれる．音声入力デバイスを追加するプラグインや，入力信号の後処理プラグイン結果出力のプラグインなどが作成できる．本章ではプラグインの動作環境，使用方法，および作成方法について，例を交えながら概要を述べる．

なお，サンプルのプラグインのソースコードが Julius のアーカイブの `plugins` 以下にある．各関数の具体的な仕様をソース上にコメントとして詳細に記述してあるので，各関数の詳細についてはそちらを参照されたい．

11.1 動作環境

`dlopen()` 関数をサポートする環境，および Windows にて動作する．`dlopen()` は Linux, Solaris など主要な OS でサポートされている．

11.2 使用方法

Julius 用プラグインのファイル拡張子は ".jpi" である．プラグインを使用するには，オプション "`-plugindir` *`dirlist`*" を指定する．*`dirlist`* には，使用したい .jpi ファイルの置いてあるディレクトリを指定する（そのディレクトリにあるすべての jpi ファイルが読み込まれる）．なお，ディレクトリが複数ある場合，*`dirlist`* にコロンで区切って並べて指定することができる．

なお，音声入力プラグインや特徴量入力プラグインなどは，他の起動時オプション ("`-input`" 等) を拡張する．このため，この"`-plugindir`" オプションはできるだけ最初のほうで指定するのがよい．

11.3 プログラミング例

C 言語によるプラグインの簡単なプログラム例，およびコンパイルの方法を以下に示す．

11.3.1 例１：認識結果出力プラグイン

この最も簡単なプラグインは，認識が完了するたびに一位の認識候補の文字列を標準出力に出力する．また，認識失敗や入力棄却の場合は `[failed]` と出力する．

Example 11.1 test1.c: a simple output plugin

```
#include <stdio.h>
#include <string.h>
int
get_plugin_info(int opcode, char *buf, int buflen)
{
  switch(opcode) {
  case 0:
    strncpy(buf, "simple output plugin", buflen);
    break;
  }
  return 0;
}
void
result_best_str(char *result_str)
{
  if (result_str == NULL) {
    printf("\t[failed]\n");
  } else {
    printf("\t[%s]\n", result_str);
  }
}
```

ここでは2つの関数を外部参照可能なように定義している．get_plugin_info は，このプラグインに関する情報を返す関数である．これは，プラグインごとに必ず定義する必要がある．result_best_str が処理の本体である．この名前の関数がプラグイン内で定義されていると，Julius は，認識結果が得られるたびにこの関数を呼びだす．result_best_str に対しては，引数として認識結果の文字列（認識失敗時，入力棄却時は NULL）が与えられる．

Julius 用プラグインファイル (.jpi) の実体は，共有オブジェクトである．例えば gcc では，以下のようにコンパイルすることができる．

```
% gcc -shared -o test.jpi test.c
```

使用するには，前述のように，Julius 起動時にオプション-plugindir でそのプラグインの置いてあるディレクトリを指定する．例えばカレントディレクトリにある場合は以下のようにする．-C ... 以降は通常の起動と同じである．

```
% julius -plugindir . -C ...
```

上記を実行すれば，認識結果が出るたびに上記の result_best_str が呼ばれ，認識結果が出力されることが確かめられる．

11.3.2 例２：音声入力プラグイン

2つ目の例として，音声入力を拡張するプラグインを解説する．音声入力プラグインで定義するのは以下の関数である．

- get_plugin_info(): プラグインの情報を返す
- adin_get_optname(): "-input" オプション用文字列を返す
- adin_get_configuration(): このプラグインのプロパティを返す
- adin_standby(): 音声入力デバイスを準備する
- adin_open(): 音声入力デバイスを開く
- adin_read(): デバイスから音声データを読み込む
- adin_close(): 音声入力デバイスを閉じる

adin_get_optname() は，このプラグインを使用して音声入力を行う際に Julius の起動時オプション -input で指定すべき文字列を指定する．

adin_get_configuration() は，この音声入力プラグイン使用時に Julius がどう音声入力をハンドルすべきかを返す．リアルタイム認識を行うか，無音区間検出を行うか，音声入力部をスレッド化すべきかなどの情報を返す．

adin_standby() は，起動時に一回呼ばれるので，デバイスのチェックなどをここで行える．adin_open() でデバイスを開き，adin_read() でバッファにあるデータを読み込み，adin_close() でデバイスを閉じる．

ヘッダ "plugin_defs.h" は julius のアーカイブの plugins ディレクトリ以下にあり，boolean などいくつかの必要な定義が含まれている．音声入力プラグイン用のいくつかの関数で，返り値や引数に用いているのでインクルードする必要がある．

Linux で OSS API を使用してマイクから音声入力を行うプラグインの例を以下に示す．[1]

[1] OSS API での入力は Julius にすでに実装されている．ここに示すのは等価なプラグイン版である．

Example 11.2 test2.c: OSS A/D-in plugin

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/soundcard.h>
#include "plugin_defs.h"

static int audio_fd;        /* audio file descpritor */
static int freq;            /* given sampling frequency */

/* return plugin information */
int
get_plugin_info(int opcode, char *buf, int buflen)
{
  switch(opcode) {
  case 0:
    strncpy(buf, "OSS adin plugin", buflen);
    break;
  }
  return 0;
}

/* return argument string for "-input" */
void
adin_get_optname(char *buf, int buflen)
{
  strncpy(buf, "myadin", buflen);
}

/* return property of this adin input */
int
adin_get_configuration(int opcode)
{
  switch(opcode) {
  case 0: /* enable real-time processing of 1st pass by default? */
    return 1;   /* yes */
  case 1:       /* enable frontend voice detection by level/zc by default? */
    return 1;   /* yes */
  case 2:       /* input module threading is needed or not, if supported? */
    return 1;   /* yes, needed*/
  }
}

/* standby, will be called once at startup */
boolean
adin_standby(int sfreq, void *dummy)
{
  freq = sfreq;  /* just store required sampling frequency to local */
  return TRUE;
}

/* open device */
boolean
adin_open(char *pathname)
{
  int fmt;
  int stereo;
  int ret;
  int s;

  if ((audio_fd = open(pathname ? pathname : "/dev/dsp", O_RDONLY)) == -1) {
    printf("Error: cannot open %s\n", pathname ? pathname : "/dev/dsp");
    return FALSE;
  }
  fmt = AFMT_S16_LE;               /* 16bit signed (little endian) */
  if (ioctl(audio_fd, SNDCTL_DSP_SETFMT, &fmt) == -1) {
    printf("Error: failed set format to 16bit signed\n");
```

コンパイルは例１と同様に行う．plugin_defs.h を Julius の plugins ディレクトリからカレントにコピーしておくこと．

```
% gcc -shared -o test2.jpi test2.c
```

使用するには，このプラグインをロードした上で，起動時に"-input" でこの音声入力プラグインを選択する．この例では関数 adin_get_optname は "myadin" をバッファに格納して返しているので，以下のようにすればこの音声入力プラグインを使った音声入力が行える．

```
% julius -plugindir . -C ... -input myadin
```

11.3.3 例３：認識開始・終了の検知プラグイン

3 つ目の例は，認識処理の開始・終了時に何らかの処理を挟み込むプラグインである．これは，Julius のコアライブラリである JuliusLib のコールバック機能を使うことで実現できる．

プログラム例を以下に示す．コールバック機能を使う場合，JuliusLib ライブラリとリンクする必要があるため，julius/juliuslib.h をインクルードしている．

Example 11.3 test3.c: Start / stop notification plugin

```
#include <julius/juliuslib.h>
int
get_plugin_info(int opcode, char *buf, int buflen)
{
  switch(opcode) {
  case 0:
    strncpy(buf, "start/stop notify plugin", buflen);
    break;
  }
  return 0;
}
static void
func_begin(Recog *recog, void *dummy)
{
  printf("[BEGIN]\n");
}
static void
func_end(Recog *recog, void *dummy)
{
  printf("[END]\n");
}
int
startup(void *data)
{
  Recog *recog = data;
  callback_add(recog, CALLBACK_EVENT_RECOGNITION_BEGIN, func_begin, NULL);
  callback_add(recog, CALLBACK_EVENT_RECOGNITION_END, func_end, NULL);
  return 0;
}
```

これまでの例と同様に，get_plugin_info は必ず定義する必要がある．2 つのローカルな関数 func_begin と func_end は，コールバックに登録する関数であり，認識処理中にそれぞれ音声認識処理の開始・終了のタイミングで呼び出される．

startup という名前の関数がプラグインにおいて定義・エクスポートされているとき，その関数は Julius の起動が完了した時点で 1 回だけ呼ばれる．呼び出しの際，引数としてエンジンインスタンス recog が与えられるので，ここではそのエンジンに対して上記のコールバックを登録している．

本プラグインは JuliusLib を使用している．JuliusLib を用いたコンパイルは以下のように行う（システムに Julius がインストールされている場合）．

```
% gcc -shared -o test3.jpi `libjulius-config --cflags` `libsent-config --cflags` \
 test3.c `libjulius-config --libs` `libsent-config --libs`
```

システム上のものではなく，バイナリパッケージ内になるライブラリと直接リンクすることもできる．この場合は以下のようにする．なお，パッケージの展開ディレクトリを $JDIR とする．

```
% gcc -shared -o test3.jpi -I$JDIR/libjulius/include -I$JDIR/libsent/include \
 `$JDIR/bin/libjulius-config --cflags` `$JDIR/bin/libsent-config --cflags` \
 test3.c \
 -L$JDIR/libjulius `libjulius-config --libs` -L$JDIR/libsent `libsent-config --libs`
```

このようにして，JuliusLib のコールバック機能をプラグインから利用できる．コールバックの一覧や使い方，引数の説明については，JuliusLib の章を参照のこと．なお，例２で使用したヘッダ "plugin_defs.h" の内容は"julius/juliuslib.h" に含まれているので，JuliusLib を使用する際は"plugin_defs.h" はインクルードしなくてよい．

11.3.4 例４：オプションを拡張する

Julius 起動時に特定のプラグインに対してパラメータを与えたい場合，以下の要領でプラグイン用の起動時オプションを追加することができる．前節の test3.c を，起動時オプション "-notify" をつけたときのみ有効にするよう拡張したプログラム test4.c を以下に示す．なお, test3.c からは，関数 opt_notify, initialize が追加され，startup が変更されている．

Example 11.4 test4.c: extend test3.c by adding an option "-notify"

```
#include <julius/juliuslib.h>
static int notify_flag = 0;

int
get_plugin_info(int opcode, char *buf, int buflen)
{
  switch(opcode) {
  case 0:
    strncpy(buf, "start/stop notify plugin with option -notify", buflen);
    break;
  }
  return 0;
}
static void
func_begin(Recog *recog, void *dummy)
{
  printf("[BEGIN]\n");
}
static void
func_end(Recog *recog, void *dummy)
{
  printf("[END]\n");
}

static boolean
opt_notify(Jconf *jconf, char *arg[], int argnum)
{
  notify_flag = 1;
  return TRUE;
}
int
initialize()
{
  j_add_option("-notify", 0, 0, "enable notify extension", opt_notify);
  return 0;
}

int
startup(void *data)
{
  Recog *recog = data;
  if (notify_flag == 1) {
    callback_add(recog, CALLBACK_EVENT_RECOGNITION_BEGIN, func_begin, NULL);
    callback_add(recog, CALLBACK_EVENT_RECOGNITION_END, func_end, NULL);
  }
  return 0;
}
```

関数 initialize が定義されている場合，それはプラグインが読み込まれた直後に実行される．initialize 関数内で呼び出している j_add_option は JuliusLib の関数であり，Julius にオプションを追加するものである．ここではオプション "-notify" が指定されたとき opt_notify 関数を呼び出すよう登録している．このプログラムでは，オプションが指定されたかどうかを static なグローバル変数 notify_flag に保存しておき，startup 内でその値が 1 のときにのみコールバックを登録するようにしている．

この test4.c をコンパイル後，test3.jpi を削除して以下を実行し，オプション "-notify" をつける場合とつけない場合でどうなるか，確かめるとよい．

```
% julius -plugindir . -C ...  -notify
```

11.4 プラグインの仕様

現時点の Julius において，定義できる関数はおおまかに以下の種類に分類できる．

- 音声入力プラグイン
- 音声後処理プラグイン
- 特徴量入力プラグイン
- 特徴量後処理プラグイン
- ガウス分布計算プラグイン
- 結果取得プラグイン
- 初期化・処理開始時など汎用関数

以下にプラグインごとの概要および定義すべきエクスポート関数の一覧を示す．なお，関数の型や引数の意味など，各関数の具体的な仕様については，julius のソースアーカイブの `plugin` ディレクトリ以下にあるプラグインのサンプルプログラムを参照のこと．

11.4.1 音声入力プラグイン

音声入力プラグインは，Julius の音声入力デバイスを拡張する．あるデバイス用のプラグインを作成すれば，Julius はそのデバイスから音声を直接取り込んで認識することができる．

このプラグインは，Julius のオプション `-input` を拡張する．`-input` につづけてプラグイン内の `adin_get_optname` 関数で返す文字列を指定することで，Julius はこのプラグインを音声入力ハンドラとして選択し，このプラグイン関数を使って音声の取り込みを行う．

音声入力プラグインで定義すべきエクスポート関数は以下の通り．

```
  - int get_plugin_info(int opcode, char *buf, int buflen)
  - int adin_get_optname(char *buf, int buflen)
  - int adin_get_configuration(int opcode)
  - boolean adin_standby(int sfreq, void *dummy)
  - boolean adin_open(char *pathname)
  - int adin_read(SP16 *buf, int sampnum)
  - boolean adin_close()
任意（省略可能）：
  - boolean adin_terminate()
  - boolean adin_pause()
  - boolaen adin_resume()
```

11.4.2 音声後処理プラグイン

音声の後処理プラグインは，Julius が取り込んだ音声信号に対して後処理を行えるプラグインである．このプラグインは，音声信号を取り込んだ直後，特徴量抽出や認識処理などの処理を行う前に呼び出される．このプラグインを使うことで，例えば入力音声信号をモニタしたり，信号に対して認識処理の前段で何らかの処理を加えることができる．呼び出す対象としては，音声区間検出前を含むすべての音声入力か，あるいは振幅と零交差数によってトリガした音声入力区間のみかを選択できる．

音声の後処理プラグインで定義すべきエクスポート関数は以下の通り．

```
  - get_plugin_info()
  - adin_postprocess() または adin_postprocess_triggered()
```

11.4.3 特徴量入力プラグイン

特徴量入力プラグインは，特徴量入力を拡張するプラグインである．これを用いることで，Julius に対して外部から特徴量ベクトル列を与えることができる．与えられた特徴量ベクトルはフレーム単位で逐次処理され，リアルタイム認識が可能である．

このプラグインはオプション `-input` を拡張する．`-input` につづけて `fvin_get_optname` で返される文字列を指定することで，Julius はこのプラグインを特徴量の入力ハンドラとして用いて認識を行う．

特徴量入力プラグインで定義すべきエクスポート関数は以下の通り．

```
  - get_plugin_info()
  - fvin_get_optname()
  - fvin_get_configuration()
  - fvin_standby()
  - fvin_open()
  - fvin_read()
  - fvin_close()
任意（省略可能）：
  - fvin_terminate()
  - fvin_pause()
  - fvin_resume()
```

なお，特徴量入力プラグインは，特徴量が一種類の場合のみ動作する．複数音響モデルを使用した認識で，各モデルが 2 つ以上の異なる特徴量を用いている場合，特徴量入力プラグインは正しく動作しない．

11.4.4 特徴量後処理プラグイン

特徴量後処理プラグインは，Julius が抽出した（あるいは外部から与えられた）特徴量ベクトルに対して後処理を行えるプラグインである．このプラグインは，入力フレームごとに，特徴量を算出した後認識を行う直前に呼ばれる．このプラグインを使うことで，特徴量ベクトルを外部へ取り出したり，認識前に加工を加えることができる．

特徴量後処理プラグインで定義すべきエクスポート関数は以下の通り．

```
  - get_plugin_info()
  - fvin_postprocess()
```

なお，特徴量後処理プラグインは，特徴量が一種類の場合のみ動作する．複数音響モデルを使用した認識で，各モデルが 2 つ以上の異なる特徴量を用いている場合，このプラグインは正しく動作しない．

11.4.5 ガウス分布計算プラグイン

ガウス分布計算プラグインは，音響モデルの出力確率（尤度）計算を行うプラグインである．使用時，この関数は認識処理においてデコーダ内部関数の代わりに呼ばれる．これを使うことで，Julius の音響尤度計算部分を任意のものに差し替えることができる．呼び出し時にはガウス分布集合と入力ベクトルが与えられるので，各ガウス分布における入力ベクトルの出力確率をそれぞれ計算して格納して返すよう作成する．

このプラグインはオプション `-gprune` を拡張する．これに `calcmix_get_optname` で返される文字列を指定することで，Julius はこのプラグインをガウス分布計算モジュールとして用いる．

このガウス分布計算プラグインで定義すべきエクスポート関数は以下の通り．

```
  - get_plugin_info()
  - calcmix_get_optname()
  - calcmix()
  - calcmix_free()
  - calcmix_init()
```

11.4.6 結果取得プラグイン

以下の関数は，認識結果の取得に使えるプラグイン関数である．入力の認識が終わるたびに呼び出され，入力に対する認識結果（最も確率の高い候補）の文字列が渡される．複数モデル認識の場合は，全ての認識処理での結果の中で最もスコアの高いものが渡される．

```
  - result_best_str()
```

このプラグインに渡される認識結果はテキスト（単語列）のみである．時間のアラインメントや競合候補などの詳細な結果を取得するには，コールバック関数を使用する．

11.4.7 その他のプラグイン関数

以下の関数は，定義しておけばそれぞれのタイミングで呼び出される汎用関数である．プラグインの初期化やコールバックの設定などに用いることができる．

```
  - initialize()
  - startup()
```

11.5 複合プラグインについて

一つのプラグインに異なる種類のプラグイン関数を書くことで，単一のプラグインに複数の機能を持たせることができる．例えば，音声入力プラグインの関数と結果取得プラグインの関数の両方を定義すれば，入力と出力を一度に拡張するプラグインを作成することができる．

11.6 制限

特徴量に関するプラグインは，複数の種類の特徴量を用いた同時認識の場合，動作しないことがある．音響モデルが 1 つであるか，または複数の音響モデルが同一の特徴量を使う場合，特徴量に関するプラグインは動作する．しかし，複数の音響モデルがそれぞれ異なる特徴量を使用する条件では正しく動作しない．

Julius のプラグイン機能全体を無効化したい場合は，コンパイル時の configure オプションで`--disable-plugin` を指定する．

Chapter 12

JuliusLib

3. Julius-simple.c を順に解説 4. コールバック一覧，処理フローとの関係 5. フォーラムへの誘導
Julius-4 では本体がライブラリとなったのでアプリに組み込める．

12.1 JuliusLib の構成と仕組み

ライブラリ，処理は基本的にコールバック

12.2 JuliusLib を用いたコンパイル

libjulius と libsent がある．
組み込み手順

12.3 julius-simple.c 解説

簡単な例で，実際に組み込んでみる．
コールバック関数の書き方，例を挙げながら．
コントロール関数の使い方，タイミングなど
jconf オプションの拡張方法について
諸注意，はまりやすいポイントなど

12.4 JuliusLib API (ver.xx)

12.4.1 処理フロー

処理フロー

12.4.2 Functions

コントロール関数一覧

12.4.3 Callbacks

コールバック一覧

12.5 Web Forum

Appendix A

バージョンごとの主な変更点

Julius の主要なリリース間の主な変更点を以下にまとめる．リリースごとの詳細な変更点については，配布アーカイブ内の変更履歴一覧 `Release-ja.txt` を参照のこと．

A.1 バージョン 4.0 から 4.1 への変更点

- プラグインのサポート
- 音響モデルにおけるマルチストリームのサポート
- MSD-HMM 音響モデルのサポート
- CVN, VTLN のサポート (`-cvn`, `-vtln`)
- `-fallback1pass` オプションの追加：バージョン 4 では，第 2 パス失敗時には認識失敗（出力無し）となっている．このオプションを指定することで，3.x と同様に，失敗時は第 1 パスの結果を最終結果とするようになる．
- Linux のオーディオ API のデフォルトを OSS から ALSA に変更
- Linux で音声入力デバイスとして ALSA, OSS, ESounD が実行時に選択可能になった：`-input alsa`, `oss`, `esd`
- バグ修正：`-multigramout`，jconf 内での環境変数展開，`-record`，その他多くの細かい修正と特定条件における動作の改善．
- MFCC 計算でエネルギー項にパワーを使うオプション追加：`-usepower`
- この文書 (The JuliusBook) の作成

A.2 バージョン 3.5.3 から 4.0 への変更点

互換性に関する変更点：

- Julian が Julius に統合された．使用方法に変更点は無く，これまで Julian に与えていたものと同じオプションを与えればほぼ等価に動作する．
- 単語グラフ出力が実行時オプションになった (`-lattice`)
- ショートポーズセグメンテーションが実行時オプションになった (`-spsegment`)．また，ポーズに対応する音響モデルのリストを指定可能に (`-pausemodels`)
- マルチパスモードが実行時オプションになった (`-multipath`)．また，音響モデルの型を判別して自動的にマルチパスモードに切り替えるようになった．
- モジュールモードの拡張：出力に `<STARTRECOG>`, `<ENDRECOG>` を追加，入力に `GRAMINFO` および認識処理プロセス制御関連を追加．

- 辞書において，第 2 項（出力文字列）の省略が可能になった．省略した場合，出力文字列は第 1 項目と同じとみなされる．これでほぼ HTK と同じ仕様になった．
- 辞書の第 1 カラムでダブルクォートが使えるようになった．

新機能：

- 複数モデル認識 (`-AM`, `-LM`, `-SR`, `-AM_GMM`, `-inactive`）
- 認識結果を入力ファイルごとに保存 (`-outfile`)
- ログをファイルに出力 / ログ出力を止める (`-logfile` / `-nolog`)
- jconf 内で環境変数を利用可能（`$VARNAME`）
- 48kHz 取り込みし 16kHz へダウンサンプリングしながら認識 (`-48`)
- マイク入力における遅延幅の調整：環境変数 `LATENCY_MSEC`
- ALSA 入力で録音デバイスを変更：環境変数 `ALSADEV`
- 平均パワーによる入力棄却 (`-powerthres`, `--enable-power-reject`
- GMM ベースの VAD (`--enable-gmm-vad`, `-gmmmargin`, `-gmmup`, `-gmmdown`)
- デコーダベース VAD (`--enable-decoder-vad -spdelay`)
- ショートポーズセグメンテーションにおいて無音に対応する音響モデルのリストの指定 (`-pausemodels`)
- 4-gram 以上の N-gram サポート
- 前向き・逆向きどちらかの N-gram のみでも動作
- ユーザ定義言語制約 (`-userlm` および関数定義)
- 辞書のみによる孤立単語認識のサポート (`-w`, `-wlist`, `-wsil`)
- Confusion network 出力 (`-confnet`)

A.3 バージョン 3.5 から 3.5.3 への変更点

- 性能改善：20%〜40%の高速化，メモリ管理の大幅な改善，Windows での動作改善
- 文法ツールの拡充：dfa_minimize, dfa_determinize の追加, HTK SLF からの文法変換ツール slf2dfa の公開
- 特徴量抽出の拡大：設定パラメータの大幅追加，MAP-CMN 実装，およびオンラインでのエネルギー項正規化のサポート
- 特徴量パラメータの HTK Config からの読み込みとバイナリファイル埋め込み機能

A.4 バージョン 3.4.2 から 3.5 への変更点

- GMM による入力棄却
- 単語グラフ出力
- 複数文法認識の正式サポート：`-multigramout`, `-gram`, `-gramlist`
- 文字コード変換：`-charconv`
- Linux (OSS) で入力デバイス変更：環境変数 `AUDIODEV`
- 圧縮ファイルの展開に zlib を使用
- 全派生版のソース統合，mingw サポート
- Doxygen によるソースコードドキュメント対応

Appendix B

オプション一覧

Julius および JuliusLib コアエンジンの設定（動作選択，設定，モデル指定，パラメータ変更など）は，すべてここで説明する「オプション」で指定する．Julius に対しては，これらのオプションをコマンドライン引数として直接指定するか，あるいはテキストファイル内に記述したものを "-C" につづけて指定する．このオプションを記述したテキストファイルは"jconf 設定ファイル" と呼ばれる．

JuliusLib を用いる他のアプリケーションにおいても，JuliusLib 内の認識エンジンへの動作指定は，同様にこのオプション群で行う．jconf 設定ファイルに設定内容を記述して，それをメイン関数の最初で `j_config_load_file_new(char *jconffile);` で呼び出すことで，JuliusLib 内の認識エンジンに設定をセットすることができる．

なお，jconf 設定ファイル内では，相対ファイルパスはその jconf ファイルの位置からの相対パスとして解釈される（カレントディレクトリではない）．注意されたい．

以下に各オプションを解説する．

B.1 アプリケーション

JuliusLib とは独立した，アプリケーションとしての Julius の機能に関するオプションである．認識結果の出力，文字コード変換，ログの設定，モジュールモードなどを含む．これらのオプションは，JuliusLib を組み込んだ他のアプリケーションでは使用できないので注意すること．

-outfile　認識結果を個別のファイルに出力する．入力ファイルごとの認識結果を，その拡張子を ".out" に変えたファイルに保存する． (rev. 4.0)

-separatescore　認識結果で言語スコアと音響スコアを個別に出力する．指定しない場合，和の値が認識結果のスコアとして出力される．

-callbackdebug　コールバックが JuliusLib から呼ばれたときにコールバック名を標準出力に出力する．デバッグ用である．(rev.4.0)

-charconv ***from to*** 出力で文字コードの変換を行う．*from* は言語モデルの文字セットを，*to* は出力の文字セットを指定する．文字セットは，Linux では `iconv` で用いられるコード名である．Windows では，コードページ番号あるいはいかに示すコード名のどれかである：`ansi`, `mac`, `oem`, `utf-7`, `utf-8`, `sjis`, `euc`.

-nocharconv　文字コード変換を行わない．`-charconv` の指定をリセットする．

-module [port] Julius を「サーバモジュールモード」で起動する．TCP/IP 経由でクライアンとやりとりし，処理の制御や認識結果・イベントの通知が行える．`port` はポート番号であり，省略時は 10500 が用いられる．

-record ***dir*** 区間検出された入力音声をファイルに逐次保存する．*dir* は保存するディレクトリを指定する．ファイル名は，それぞれの処理終了時のシステム時間から `YYYY.MMDD.HHMMSS.wav` の形で保存される．ファイル形式は 16bit, 1 チャンネルの WAV 形式である．なお，入力が GMM 等によって棄却された場合も記録される．

-logfile ***file*** 通常 Julius は全てのログ出力を標準出力に出力する．このオプションを指定することで，それらの出力を指定ファイルに切替えることができる．(Rev.4.0)

-nolog ログ出力を禁止する．標準出力には何も出力されなくなる．(Rev.4.0)

-help エンジン設定，オプション一覧などのヘルプを出力して終了する．

B.2 全体オプション

全体オプションは，モデルや探索以外のオプションであり，音声入力・音検出・GMM・プラグイン・その他の設定を含む．全体オプションは，音響モデル (-AM)・言語モデル (-LM)・デコーダ (-SR) などのセクション定義の前に定義するか，-GLOBAL のあとに指定する．

B.2.1 オーディオ入力

-input {mic|rawfile|mfcfile|adinnet|stdin|netaudio|esd|alsa|oss} 音声入力ソースを選択する．音声波形ファイルの場合は file あるいは rawfile を指定する．HTK 形式の特徴量ファイルを認識する場合は htkparam あるいは mfcfile を指定する．起動後にプロンプトが表れるので，それに対してファイル名を入力する．adinnet では，adintool などのクライアントプロセスから音声データをネットワーク経由で受け取ることができる．netaudio は DatLink のサーバから，stdin は標準入力からの音声入力を認識する．esd は，音声デバイスの共有手段として多くの Linux のデスクトップ環境で利用されている EsounD daemon からの入力を認識する．

-filelist *filename* (-input rawfile|mfcfile 時) *filename* 内に列挙されている全てのファイルについて認識を順次行う．*filename* には認識する入力ファイル名を 1 行に 1 つずつ記述する．

-notypecheck 入力の特徴量ベクトルの型チェックを無効にする．通常 Julius は入力の型が音響モデルとマッチするかどうかをチェックし，マッチしないとエラー終了する．このオプションはそのチェックを回避する．なんらかの理由で型チェックがうまく動作しないときに使用する．

-48 48kHz で入力を行い，16kHz にダウンサンプリングしながら認識する．これは 16kHz のモデルを使用しているときのみ有効である．ダウンダンプリングの内部機能は sptk から移植された．(Rev. 4.0)

-NA *devicename* DatLink サーバのデバイス名 (-input netaudio).

-adport *port_number* -input adinnet 使用時，接続を受け付ける adinnet のポート番号を指定する．(default: 5530)

-nostrip 音声取り込み時，デバイスやファイルによっては，音声波形中に振幅が "0" となるフレームが存在することがある．Julius は通常，音声入力に含まれるそのようなフレームを除去する．この零サンプル除去がうまく動かない場合，このオプションを指定することで自動消去を無効化することができる．

-zmean ，-nozmean 入力音声ストリームに対して直流成分除去を行う．全ての音声処理のの前段として処理される．-zmeansource オプションも見よ．

B.2.2 レベルと零交差による入力検知

-cutsilence ，-nocutsilence レベルと零交差による入力検知を行うかどうかを指定する．デフォルトは，リアルタイム認識（デバイス直接入力およびネットワーク入力）では on, ファイル入力では off である．このオプションを指定することで，例えば長時間録音された音声ファイルに対して音声区間検出を行いながら認識を行うこともできる．

-lv *thres* 振幅レベルのしきい値．値は 0 から 32767 の範囲で指定する．(default: 2000)

-zc *thres* 零交差数のしきい値．値は 1 秒あたりの交差数で指定する．(default: 60)

-headmargin *msec* 音声区間開始部のマージン．単位はミリ秒．(default: 300)

-tailmargin *msec* 音声区間終了部のマージン．単位はミリ秒．(default: 400)

B.2.3 入力棄却

入力長，あるいは平均パワーによる入力の事後棄却が行える．平均パワーによる棄却は，デフォルトでは無効化されており，ソースからコンパイルする際に `configure` に`--enable-power-reject` を指定することで有効となる．リアルタイム認識時で，かつ特徴量でパワー項を持つ場合のみ使用できる．

-rejectshort ***msec*** 検出された区間長が *msec* 以下の入力を棄却する．その区間の認識は中断・破棄される．

-powerthres ***thres*** 切り出し区間の平均パワーのしきい値．(Rev.4.0)

このオプションはコンパイル時に`--enable-power-reject` が指定されたときに有効となる．

B.2.4 GMM / GMM-VAD

-gmm ***hmmdefs_file*** GMM 定義ファイル．3 状態（出力状態が 1 つのみ）の HMM として定義する．形式は HTK 形式で与える．形式や使用できる特徴量の制限は音響モデルと同じである．なお，GMM で用いる MFCC 特徴量の設定は，`-AM_GMM` のあとに音響モデルと同様に指定する．この特徴量設定は音響モデルと別に，明示的に指定する必要があることに注意が必要である．

-gmmnum ***number*** GMM 指定時，計算するガウス分布数を指定する．フレームごとに GMM の出力確率を求める際，各モデルで定義されている混合ガウス分布のうち，この *number* で指定した数の上位ガウス分布の確率のみを計算する．小さな値を指定するほど GMM の計算量を削減できるが，計算精度が悪くなる．(default: 10)

-gmmreject ***string*** GMM で定義されているモデル名のうち，非音声として棄却すべきモデルの名称を指定する．モデル名を複数指定することができる．複数指定する場合は，空白を入れずコンマで区切って一つの *string* として指定する．

-gmmmargin ***frames*** (GMM_VAD) GMM VAD による区間検出の開始部マージン．単位はフレーム数で指定する．(default: 20) (Rev. 4.0)

このオプションは`--enable-gmm-vad` 付きでコンパイルされたときに有効となる．

-gmmup ***value*** (GMM_VAD) 音声区間の開始とみなす VAD スコアの閾値．VAD スコアは (音声 GMM の最大尤度 - 非音声 HMM の最大尤度) で表される．(Default: 0.7) (Rev.4.1)

このオプションは`--enable-gmm-vad` 付きでコンパイルされたときに有効となる．

-gmmdown ***value*** (GMM_VAD) 音声区間の終了とみなす VAD スコアの閾値．VAD スコアは (音声 GMM の最大尤度 - 非音声 HMM の最大尤度) で表される．(Default: -0.2) (Rev.4.1)

このオプションは`--enable-gmm-vad` 付きでコンパイルされたときに有効となる．

B.2.5 デコーディング

デコーディングオプションは，使用する認識アルゴリズムに関する設定を行うオプションである．この設定はエンジン全体に対する設定であり，全ての認識処理インスタンスで共通の設定となる．探索の幅や言語重みなどの個々のデコーディング設定については，認識処理インスタンスごとに指定する．

-realtime , -norealtime 入力と並行してリアルタイム認識を行うかどうかを明示的に指定する．デフォルトの設定は入力デバイスに依存し，マイクロフォン等のデバイス直接認識，ネットワーク入力，および DatLink/NetAudio 入力の場合は ON, ファイル入力や特徴量入力については OFF となっている．

B.2.6 その他

-C ***jconffile*** jconf 設定ファイルを読み込む．ファイルの内容がこの場所に展開される．

-version バージョン情報を標準エラー出力に出力して終了する．

-setting エンジン設定情報を標準エラー出力に出力して終了する．

-quiet 出力を抑制する．認識結果は単語列のみが出力される．

-debug (デバッグ用) モデルの詳細や探索過程の記録など，様々なデバッグ情報をログに出力する．

-check {wchmm|trellis|triphone} デバッグ用のチェックモードに入る．

-plugindir *dirlist* プラグインを読み込むディレクトリを指定する．複数の場合はコロンで区切って並べて指定する．

B.3 インスタンス宣言

-AM *name* 音響モデルインスタンスを新たに宣言する．以降の音響モデルに関する設定はこのインスタンスに対するものと解釈される．*name* にはインスタンスにつける名前を指定する（既にある音響モデルインスタンスと同じ名前であってはいけない）. (Rev.4.0)

-LM *name* 言語モデルインスタンスを新たに宣言する．以降の言語モデルに関する設定はこのインスタンスに対するものと解釈される．*name* にはインスタンスにつける名前を指定する（既にある言語モデルインスタンスと同じ名前であってはいけない）. (Rev.4.0)

-SR *name am_name lm_name* 認識処理インスタンスを新たに宣言する．以降の認識処理や探索に関する設定はこのインスタンスに対するものと解釈される．*name* にはインスタンスにつける名前を指定する（既にある認識処理インスタンスと同じ名前であってはいけない）．*am_name*, *lm_name* にはそれぞれこのインスタンスが使用する音響モデルと言語モデルのインスタンスを名前，あるいは ID 番号で指定する．(Rev.4.0)

-AM_GMM GMM 使用時に，GMM 計算のための特徴量抽出パラメータを，この宣言のあとに指定する．もし GMM 使用時にこのオプションで GMM 用の特徴量パラメータを指定しなかった場合，最後に指定した音響モデル用の特徴量がそのまま用いられる．(Rev.4.0)

-GLOBAL 全体オプション用のセクションを開始する．-AM, -LM, -SR などのインスタンス宣言を用いる場合，音声入力設定などの全体オプションは，これらの全てのインスタンス定義よりも前か，あるいはこのオプションのあとに指定する必要がある．この全体オプション用のセクションは，jconf 内で何回現れても良い．(Rev.4.1)

-nosectioncheck , -sectioncheck 複数インスタンスを用いる jconf において，オプションの位置チェックの有効・無効を指定する．有効である場合，ある種類のインスタンスの宣言がされたあとは，他のインスタンス宣言が現れるまで，そのインスタンスのオプションしか指定できない（例：-AM のあと，他の -AM や-LM などが現れるまで，音響モデルオプションしか指定できない．他のオプションがあらわれた場合はエラーとなる）．また，全体オプションは全てのモデルインスタンスの前に指定する必要がある．デフォルトでは有効になっている．(Rev.4.1)

B.4 言語モデル (-LM)

このグループには，各モデルタイプごとに指定するオプションが含まれている．一つのインスタンスには一つのモデルタイプだけが指定可能である．

B.4.1 N-gram

-d *bingram_file* 使用する N-gram をバイナリファイル形式で指定する．バイナリ形式への変換は mkbingram を使用する．

-nlr *arpa_ngram_file* 前向き (left-to-right) の N-gram 言語モデルを指定する．*arpa_ngram_file* は ARPA 標準形式のファイルである必要がある．

-nrl *arpa_ngram_file* 後ろ向き (right-to-left) の N-gram 言語モデルを指定する．*arpa_ngram_file* は ARPA 標準形式のファイルである必要がある．

-v *dict_file* N-gram，または文法用の単語辞書ファイルを指定する．

-silhead *word_string* -siltail *word_string* 音声入力両端の無音区間に相当する「無音単語」エントリを指定する．単語の読み（N-gram エントリ名），あるいは"#"+単語番号（辞書ファイルの行番号-1）で指定する．デフォルトはそれぞれ "<s>", "</s>" である．

-mapunk ***word_string*** unknown に対応する単語名を指定する．デフォルトは，"<unk>" あるいは "<UNK>" である．この単語は，認識辞書において N-gram にない単語を指定した場合にマッピングされる単語である．

-iwspword ポーズに対応する無音単語を辞書に追加する．追加される単語の内容はオプション-iwspentry で変更できる．

-iwspentry ***word_entry_string*** -iwspword 指定時に追加される単語エントリの内容を指定する．辞書エントリと同じ形式で指定する．(default: "<UNK> [sp] sp sp")

-sepnum ***number*** 木構造化辞書の構築時に線形登録する単語数を指定する．(default: 150)

B.4.2 記述文法

-gram や-gramlist で文法を複数回指定することで，一つのインスタンス内で複数の文法を用いることができる．(旧 Julius のオプション -dfa, -v の組合せは単一の文法のみ指定可能である)

-gram gramprefix1[,gramprefix2[,gramprefix3,...]] 認識に使用する文法を指定する．文法はファイル（辞書および構文制約有限オートマトン）のプレフィックスで指定する．すなわち，ある認識用文法が dir/foo.dict と dir/foo.dfa としてあるとき，dir/foo のように拡張子を省いた名前で指定する．文法はコンマで区切って複数指定することができる．また繰り返し使用することでも複数指定できる．

-gramlist ***list_file*** 認識に使用する文法のリストをファイルで指定する．*list_file* には，-gram と同様の文法プレフィックスを 1 行に 1 つずつ記述する．また，このオプションを繰り返し使用することで，複数のリストファイルを指定できる．なお，リスト内で文法を相対パスで指定した場合，それは，そのリストファイルからの相対パスとして解釈されることに注意が必要である．

-dfa ***dfa_file*** **-v** ***dict_file*** 認識に使用する文法の構文制約オートマトンと辞書をそれぞれ指定する．(Julius-3.x との互換性のための古いオプションであり，使用すべきでない)

-nogram それまでに-gram，-gramlist，-dfa および -v で指定された文法のリストをクリアし，文法の指定なしの状態にする．

B.4.3 単語辞書（孤立単語認識）

-w ***dict_file*** 単単語認識で用いる単語辞書を指定する．ファイル形式は単語 N-gram や文法と同一である．辞書上の全ての単語が認識対象となる．(Rev.4.0)

-wlist ***list_file*** 単語辞書のリストを指定する．*list_file* には 1 行に一つずつ，使用する単語辞書のパスを記述する．相対パスを用いた場合，それはその *list_file* からの相対パスとして解釈される． (Rev.4.0)

-nogram それまでに-w あるいは -wlist で指定された辞書のリストをクリアし，指定なしの状態に戻す．

-wsil ***head_sil_model_name tail_sil_model_name sil_context_name*** 単単語認識時，音声入力の両端の無音モデルおよびそのコンテキスト名を指定する．*sil_context_name* として NULL を指定した場合，各モデル名がそのままコンテキストとして用いられる．

B.4.4 ユーザ定義 LM

-userlm プログラム中のユーザ定義言語スコア計算関数を使用することを宣言する．(Rev.4.0)

B.4.5 その他

-forcedict 単語辞書読み込み時のエラーを無視する．通常 Julius は単語辞書内にエラーがあった場合そこで動作を停止するが，このオプションを指定することで，エラーの生じる単語をスキップして処理を続行することができる．

B.5 音響モデル・特徴量抽出 (-AM) (-AM_GMM)

音響モデルオプションは，音響モデルおよび特徴量抽出・フロントエンド処理に関する設定を行う．特徴量抽出，正規化処理，スペクトルサブトラクションの指定もここで行う．

B.5.1 音響モデル・HMM

-h ***hmmdef_file*** 使用する HMM 定義ファイル．HTK の ASCII 形式ファイル，あるいは Julius バイナリ形式のファイルのどちらかを指定する．バイナリ形式へは mkbinhmm で変換できる．

-hlist ***hmmlist_file*** HMMlist ファイルを指定する．テキスト形式，あるいはバイナリ形式のどちらかを指定する．バイナリ形式へは mkbinhmmlist で変換できる．

-tmix ***number*** Gaussianpruning の計算状態数を指定する．小さいほど計算が速くなるが，音響尤度の誤差が大きくなる．See also `-gprune`. (default: 2)

-spmodel ***name*** 文中のショートポーズに対応する音韻 HMM の名前を指定する．このポーズモデル名は，`-iwsp`, `-spsegment`, `-pausemodels` に関係する．また，文法使用時にスキップ可能なポーズ単語エントリの識別にも用いられる．(default: "sp")

-multipath **-nomultipath** 状態間遷移を拡張するマルチパスモードを有効にする．オプション指定がない場合，Julius は音響モデルの遷移をチェックし，必要であれば自動的にマルチパスモードを有効にする．このオプションは，ユーザが明示的にモードを指定したい場合に使用する．

この機能は 3.x ではコンパイル時オプションであったが，4.0 より実行時オプションとなった．(rev.4.0)

-gprune **{safe|heuristic|beam|none|default}** 使用する Gaussian pruning アルゴリズムを選択する．`none` を指定すると Gaussian pruning を無効化しすべてのガウス分布について厳密に計算する．`safe` は上位 N 個を計算する．`heuristic` と `beam` は safe に比べてより積極的な枝刈りを行うため計算量削減の効果が大きいが，認識精度の低下を招く可能性がある．`default` が指定された場合，デフォルトの手法を使う．(default: tied-mixture model の場合，standard 設定では `safe`，fast 設定では `beam`．tied-mixture でない場合 `none`).

-iwcd1 **{max|avg|best number}** 第 1 パスの単語間トライフォン計算法を指定する．`max` 指定時，同じコンテキストのトライフォン集合の全尤度の最大値を近似尤度として用いる．`avg` は全尤度の平均値を用いる．`best number` は上位 N 個のトライフォンの平均値を用いる．デフォルトは，一緒に使用される言語モデルに依存する．N-gram 使用時には `best 3`，文法使用時は `avg` となる．もしこの音響モデルが異なるタイプの複数の言語モデルで共有される場合は，後に定義されたほうのデフォルトがデフォルト値として用いられる．

-iwsppenalty ***float*** `-iwsp` によって末尾に付加される単語末ショートポーズの挿入ペナルティ．ここで指定した値が，通常単語の末尾から単語末ショートポーズへの遷移に追加される．

-gshmm ***hmmdef_file*** Gaussian Mixture Selection 用のモノフォン音響モデルを指定する．GMS 用モノフォンは通常のモノフォンから mkgshmm によって生成できる．

-gsnum ***number*** GMS 使用時，対応するトライフォンを詳細計算するモノフォンの状態の数を指定する． (default: 24)

B.5.2 特徴量抽出

-smpPeriod ***period*** 音声のサンプリング周期を指定する．単位は，100 ナノ秒の単位で指定する．サンプリング周期は `-smpFreq` でも指定可能．(default: 625)

このオプションは HTK の `SOURCERATE` に対応する．同じ値が指定できる．

複数の音響モデルを用いる場合，全インスタンスで共通の値を指定する必要がある．

-smpFreq ***Hz*** 音声のサンプリング周波数 (Hz) を指定する．(default: 16,000)

複数の音響モデルを用いる場合，全インスタンスで共通の値を指定する必要がある．

-fsize ***sample_num*** 窓サイズをサンプル数で指定 (default: 400).

このオプションは HTK の WINDOWSIZE に対応する．ただし値は HTK と異なり，(HTK の値 / smpPeriod) となる．

複数の音響モデルを用いる場合，全インスタンスで共通の値を指定する必要がある．

-fshift ***sample_num*** フレームシフト幅をサンプル数で指定 (default: 160).

このオプションは HTK の TARGETRATE に対応する．ただし値は HTK と異なり，(HTK の値 / smpPeriod) となる．

複数の音響モデルを用いる場合，全インスタンスで共通の値を指定する必要がある．

-preemph ***float*** プリエンファシス係数 (default: 0.97)

このオプションは HTK の PREEMCOEF に対応する．同じ値が指定できる．

-fbank ***num*** フィルタバンクチャンネル数．(default: 24)

このオプションは HTK の NUMCHANS に対応する．同じ値が指定できる．指定しないときのデフォルト値が HTK と異なっていることに注意（HTK では 22）．

-ceplif ***num*** ケプストラムのリフタリング係数. (default: 22)

このオプションは HTK の CEPLIFTER に対応する．同じ値が指定できる．

-rawe , -norawe エネルギー項の値として，プリエンファシス前の raw energy を使用する / しない (default: disabled=使用しない)

このオプションは HTK の RAWENERGY に対応する．指定しないときのデフォルトが HTK と異なっていることに注意（HTK では enabled).

-enormal , -noenormal エネルギー項の値として，発話全体の平均で正規化した正規化エネルギーを用いるかどうかを指定する．(default: -noenormal)

このオプションは HTK の ENORMALISE に対応する．指定しないときのデフォルトが HTK と異なっていることに注意（HTK では enabled).

-escale ***float_scale*** エネルギー正規化時の，対数エネルギー項のスケーリング係数．(default: 1.0)

このオプションは HTK の ESCALE に対応する．デフォルト値が HTK と異なっていることに注意（HTK では 0.1）．

-silfloor ***float*** エネルギー正規化時の，無音部のエネルギーのフロアリング値．(default: 50.0)

このオプションは HTK の SILFLOOR に対応する．同じ値が指定できる．

-delwin ***frame*** 一次差分計算用のウィンドウフレーム幅．(default: 2)

このオプションは HTK の DELTAWINDOW に対応する．同じ値が指定できる．

-accwin ***frame*** 二次差分計算用のウィンドウフレーム幅．(default: 2)

このオプションは HTK の ACCWINDOW に対応する．同じ値が指定できる．

-hifreq ***Hz*** MFCC のフィルタバンク計算時におけるバンド制限を有効化する．このオプションではカットオフ周波数の上限値を指定する．-1 を指定することで無効化できる．(default: -1)

このオプションは HTK の HIFREQ に対応する．同じ値が指定できる．

-lofreq ***Hz*** MFCC のフィルタバンク計算時におけるバンド制限を有効化する．このオプションではカットオフ周波数の下限値を指定する．-1 を指定することで無効化できる．(default: -1)

このオプションは HTK の LOFREQ に対応する．同じ値が指定できる．

-zmeanframe , -nozmeanframe 窓単位の直流成分除去を有効化／無効化する．(default: disabled)

このオプションは HTK の ZMEANSOURCE に対応する．-zmean も参照のこと．

-usepower フィルタバンク解析で振幅の代わりにパワーを使う．(default: disabled)

B.5.3 正規化処理

-cvn ケプストラム分散正規化 (cepstral variance normalization; CVN) を有効にする．ファイル入力では，入力全体の分散に基づいて正規化が行われる．直接入力ではあらかじめ分散が得られないため，最後の入力の分散で代用される．音声信号入力でのみ有効である．

-vtln ***alpha lowcut hicut*** 周波数ワーピングを行う．声道長正規化 (vocal tract length normalization; VTLN) に使用できる．引数はそれぞれワーピング係数，周波数上端，周波数下端であり，HTK 設定の WARPFREQ，WARPHCUTOFF および WARPLCUTOFF に対応する．

-cmnload ***file*** 起動時にケプストラム平均ベクトルを *file* から読み込む．ファイルは-cmnsave で保存されたファイルを指定する．これは MAP-CMN において，起動後最初の発話においてケプストラム平均の初期値として用いられる．通常，2 発話目以降は初期値は，直前の入力の平均に更新されるが，-cmnnoupdate を指定された場合，常にこのファイルの値が各発話の初期値として用いられる．

-cmnsave ***file*** 認識中に計算したケプストラム平均ベクトルを *file* へ保存する．すでにファイルがある場合は上書きされる．この保存は音声入力が行われるたびに上書きで行われる．

-cmnupdate -cmnnoupdate 実時間認識時，初期ケプストラム平均を入力ごとに更新するかどうかを指定する．通常は有効 (-cmnupdate) であり，過去 5 秒間の入力の平均を初期値として更新する．-cmnnoupdate が指定された場合，更新は行われず，初期値は起動時の値に固定される．-cmnload で初期値を指定することで，常に同じ初期値を使うようにすることができる．

-cmnmapweight ***float*** MAP-CMN の初期ケプストラム平均への重みを指定する．値が大きいほど初期値に長時間依存し，小さいほど早く現入力のケプストラム平均を用いるようになる．(default: 100.0)

B.5.4 フロントエンド処理

-sscalc 入力先頭の無音部を用いて，入力全体に対してスペクトルサブトラクションを行う．先頭部の長さは-sscalclen で指定する．ファイル入力に対してのみ有効である．-ssload と同時に指定できない．

-sscalclen ***msec*** -sscalc オプション指定時，各ファイルにおいてノイズスペクトルの推定に用いる長さをミリ秒で指定する．(default: 300)

-ssload ***file*** ノイズスペクトルを *file* から読み込んでスペクトルサブトラクションを行う．*file* はあらかじめ mkss で作成する．マイク入力・ネットワーク入力などのオンライン入力でも適用できる．-sscalc と同時に指定できない．

-ssalpha ***float*** -sscalc および-ssload 用の減算係数を指定する．この値が大きいほど強くスペクトル減算を行うが，減算後の信号の歪も大きくなる．(default: 2.0)

-ssfloor ***float*** スペクトルサブトラクションのフロアリング係数を指定する．スペクトル減算時，計算の結果パワースペクトルが負となってしまう帯域に対しては，原信号にこの係数を乗じたスペクトルが割り当てられる．(default: 0.5)

B.5.5 その他

-htkconf ***file*** HTK Config ファイルを解析して，対応する特徴量抽出オプションを Julius に自動設定する．*file* は HTK で音響モデル学習時に使用した Config ファイルを指定する．なお，Julius と HTK ではパラメータのデフォルト値が一部異なるが，このオプションを使用する場合，デフォルト値も HTK のデフォルトに切替えられる．

B.6 認識処理・探索 (-SR)

認識処理・探索オプションは，第 1 パス・第 2 パス用のビーム幅や言語重みのパラメータ，ショートポーズセグメンテーションの設定，単語ラティス・CN 出力用設定，forced alignment の指定，その他の認識処理と結果出力に関するパラメータを含む．

B.6.1 第 1 パスパラメータ

-lmp ***weight penalty*** (N-gram 使用時) 第 1 パス用の言語スコア重みおよび挿入ペナルティ．ペナルティは負であれば単語挿入を抑制し，正であれば単語挿入を促進する．

-penalty1 ***penalty*** (文法使用時) 第 1 パス用の単語挿入ペナルティ． (default: 0.0)

-b ***width*** 第 1 パス探索の枝刈り (rank pruning) のビーム幅を指定する．単位は HMM ノード数である．デフォルト値は音響モデルやエンジンの設定による．モノフォン使用時は 400, トライフォン使用時は 800，トライフォンでかつ setup=v2.1 のときは 1000 となる．

-nlimit ***num*** 第 1 パスでノードごとに保持する仮説トークンの最大数．通常は 1 で固定されており変更できない．コンパイル時に `configure` で`--enable-wpair` および`--enable-wpair-nlimit` が指定されているときのみ変更できる．

-progout 第 1 パスで，一定時間おきにその時点での最尤仮説系列を出力する．

-proginterval ***msec*** `-progout` の出力インターバルをミリ秒で指定する．(default: 300)

B.6.2 第 2 パスパラメータ

-lmp2 ***weight penalty*** (N-gram 使用時) 第 2 パス用の言語スコア重みおよび挿入ペナルティ．ペナルティは負であれば単語挿入を抑制し，正であれば単語挿入を促進する．

-penalty2 ***penalty*** (文法使用時) 第 2 パス用の単語挿入ペナルティ． (default: 0.0)

-b2 ***width*** 第 2 パス探索における仮説展開回数の上限を指定する．単位は仮説数．(default: 30)

-sb ***float*** 第 2 パスの仮説尤度計算時のスコア幅を指定する．単位は対数尤度差である．(default: 80.0)

-s ***num*** 仮説のスタックサイズを指定する．(default: 500)

-n ***num*** *num* 個の文仮説数が見付かるまで探索を行う．得られた仮説はスコアでソートされて出力される（`-output` も見よ）．デフォルト値はコンパイル時のエンジン設定によって変わり，fast 版では 1, standard 版では 10 である．

-output ***num*** 見つかった N-best 候補のうち，結果として出力する文仮説の数を指定する．`-n` も参照のこと．(default: 1)

-m ***count*** 探索打ち切りのための仮説展開回数のしきい値を指定する．(default: 2000)

-lookuprange ***frame*** 第 2 パスの単語展開時に，接続しうる次単語候補を見付けるための終端時刻の許容幅をフレーム数で指定する．値を大きくするほどその周辺の多くの仮説を次単語候補として仮説展開が行われるようになるが，探索が前に進みにくくなることがある．(default: 5)

-looktrellis 仮説展開を第 1 パスの結果単語トレリス上に絞る．

B.6.3 ショートポーズセグメンテーション / デコーダ VAD

-spsegment ショートポーズセグメンテーションを有効にする． (Rev.4.0)

-spdur ***frame*** 無音区間判定のためのしきい値を指定する．無音単語が一位仮説となるフレームがこの値以上続いたとき，無音区間として入力が区切られる．(default: 10)

-pausemodels ***string*** 「無音単語」を定義するための音響モデルにおける無音モデルの名前を指定する．コンマで区切って複数の名前を指定できる．このオプションが指定されない場合，文法を用いた認識では`-spmodel` で指定されるモデルのみを読みとる単語が無音単語とされる．また，N-gram ではこれに加えて`-silhead` および `-siltail` で指定される単語も無音単語として扱われる．(Rev.4.0)

-spmargin ***frame*** デコーダベース VAD において，アップトリガ時の巻戻し幅をフレーム数で指定する．(Rev.4.0)

このオプションは `configure` に`--enable-decoder-vad` を付けてコンパイルしたときのみ有効である．

-spdelay ***frame*** デコーダベース VAD において，アップトリガ判定の遅延幅をフレーム数で指定する．(Rev.4.0)

このオプションは `configure` に`--enable-decoder-vad` を付けてコンパイルしたときのみ有効である．

B.6.4 単語ラティス / confusion network 出力

-lattice ， **-nolattice** 単語グラフ（ラティス）の出力を有効化/無効化する．

-confnet ， **-noconfnet** Confusion network の出力を有効化/無効化する．confusion network は単語グラフから生成されるため，有効時は同時に`-lattice` も有効化される．(Rev.4.0)

-graphrange ***frame*** グラフ生成において近傍の同一単語仮説をマージする．開始フレームおよび終了フレームの位置の差がそれぞれ *frame* 以下の同一単語仮説についてマージする．その際，スコアは高いほうのものが残される．値が -1 の場合，マージは一切行われない．値を大きくするほどコンパクトなグラフが生成されるが，スコアの誤差が大きくなる．このオプションは`-confnet` にも影響する．(default: 0)

-graphcut ***depth*** 生成されたグラフに対して，深さによるカットオフを行う．*depth* は，あるフレームにおいて存在可能な単語数の上限を指定する．Julius では，第 2 パスの探索が不安定な場合，一部分が極端に深いグラフが生成されることが稀にあり，このオプションによってそれを抑制することができる．-1 を指定することでこの機能は無効化される．(default: 80)

-graphboundloop ***count*** 事後的に行われる単語グラフの境界時間調整において，振動による無限ループを防ぐための打ち切り値を指定する．(default: 20)

-graphsearchdelay ， **-nographsearchdelay** 巨大グラフ生成用にアルゴリズムをチューニングする．このオプションが有効時，Julius は第 1 文仮説が見つかる前のグラフ生成時の仮説中断を行わないように，グラフ生成アルゴリズムを変更する．これは，ビーム幅や探索範囲を極端に大きくして巨大なワードグラフを生成しようとするときに，グラフの精度を改善することがある．(default: disabled)

B.6.5 複数文法認識

文法や単単語認識において，一つのインスタンスで複数の文法や辞書を用いる場合に指定できるオプションである．

-multigramout ， **-nomultigramout** 複数文法あるいは複数辞書を用いて認識を行う場合，通常の Julius は全ての文法/辞書の中から最尤仮説を出力する．このオプションを指定することで，与えられた個々の文法/辞書ごとに一位仮説を出力することができる．(default: disabled)

B.6.6 Forced alignment

-walign 認識結果を用いて，入力に対する単語単位の forced alignment を行う．単語の境界フレームと平均音響尤度が出力される．

-palign 認識結果を用いて，入力に対する音素単位の forced alignment を行う．音素ごとの境界フレームと平均音響尤度が出力される．

-salign 認識結果を用いて，入力に対する HMM の状態単位の forced alignment を行う．状態ごとの境界フレームと平均音響尤度が出力される．

B.6.7 その他

-inactive 認識処理インスタンスを一時停止状態 (inactive state) で起動する．(Rev.4.0)

-1pass 第 1 パスのみを実行する．このオプションを指定した場合，第 2 パスは実行されない．

-fallback1pass 通常，第 2 パスの探索が失敗したとき，Julius は認識結果無しで終了する．このオプションを指定することで，そのような第 2 パスの失敗時に，第 1 パスの最尤仮説を最終結果として出力することができる．（これは Julius-3.x でのデフォルトの振る舞いである）

-no_ccd , -force_ccd 音響モデルを音素コンテキスト依存モデルとして扱うかどうかを明示的に指定する．デフォルトは HMM 中のモデル名から自動判断される．

-cmalpha ***float*** 確信度計算のためのスコアのスムージング係数．(default: 0.05)

-iwsp （マルチパスモード時のみ有効）単語間にショートポーズモデルを挟み込んだ認識処理を行う．このオプションを指定すると，辞書上の全単語の末尾に，スキップ可能なショートポーズモデルが付加される．このショートポーズモデルはコンテキストを考慮せず，また前後の音素のコンテキストにも表れない．付加するショートポーズモデルは`-spmodel` で指定できる．

-transp ***float*** 透過単語に対する追加の挿入ペナルティを指定する．(default: 0.0)

-demo `-progout -quiet` と同等．

Appendix C

リファレンス・マニュアル

C.1 julius

名前

julius – 大語彙連続音声認識エンジン

Synopsis

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

内容

Julius は数万語を対象とした大語彙連続音声認識を行うことのできるフリーの認識エンジンです．単語 N-gram を用いた 2 パス構成の段階的探索により高精度な認識を行うことができます．また，小規模語彙のための文法ベースの認識や単単語認識も行うことができます．認識対象としてマイク入力，録音済みの音声波形ファイル，特徴抽出したパラメータファイルなどに対応しています．

コアの認識処理は，全て JuliusLib ライブラリとして提供されています．Julius は JuliusLib を用いる音声アプリケーションの一つです．

Julius を用いて音声認識を行うには，音響モデル，単語辞書，および言語モデルが必要です．

設定

Julius および JuliusLib コアエンジンの設定（動作選択，設定，モデル指定，パラメータ変更など）は，すべてここで説明する「オプション」で指定する．Julius に対しては，これらのオプションをコマンドライン引数として直接指定するか，あるいはテキストファイル内に記述したものを "`-C`" につづけて指定する．このオプションを記述したテキストファイルは"jconf 設定ファイル" と呼ばれる．

JuliusLib を用いる他のアプリケーションにおいても，JuliusLib 内の認識エンジンへの動作指定は，同様にこのオプション群で行う．jconf 設定ファイルに設定内容を記述して，それをメイン関数の最初で `j_config_load_file_new(char *jconffile);` で呼び出すことで，JuliusLib 内の認識エンジンに設定をセットすることができる．

なお，jconf 設定ファイル内では，相対ファイルパスはその jconf ファイルの位置からの相対パスとして解釈される（カレントディレクトリではない）．注意されたい．

以下に各オプションを解説する．

Julius アプリケーションオプション

JuliusLib とは独立した，アプリケーションとしての Julius の機能に関するオプションである．認識結果の出力，文字コード変換，ログの設定，モジュールモードなどを含む．これらのオプションは，JuliusLib を組み込んだ他のアプリケーションでは使用できないので注意すること．

`-outfile`　認識結果を個別のファイルに出力する．入力ファイルごとの認識結果を，その拡張子を ".out" に変えたファイルに保存する． (rev. 4.0)

`-separatescore`　認識結果で言語スコアと音響スコアを個別に出力する．指定しない場合，和の値が認識結果のスコアとして出力される．

-callbackdebug コールバックが JuliusLib から呼ばれたときにコールバック名を標準出力に出力する．デバッグ用である．(rev.4.0)

-charconv *from to* 出力で文字コードの変換を行う．*from* は言語モデルの文字セットを，*to* は出力の文字セットを指定する．文字セットは，Linux では iconv で用いられるコード名である．Windows では，コードページ番号あるいはいかに示すコード名のどれかである：ansi, mac, oem, utf-7, utf-8, sjis, euc.

-nocharconv 文字コード変換を行わない．-charconv の指定をリセットする．

-module [port] Julius を「サーバモジュールモード」で起動する．TCP/IP 経由でクライアンとやりとりし，処理の制御や認識結果・イベントの通知が行える．port はポート番号であり，省略時は 10500 が用いられる．

-record *dir* 区間検出された入力音声をファイルに逐次保存する．*dir* は保存するディレクトリを指定する．ファイル名は，それぞれの処理終了時のシステム時間から YYYY.MMDD.HHMMSS.wav の形で保存される．ファイル形式は 16bit, 1 チャンネルの WAV 形式である．なお，入力が GMM 等によって棄却された場合も記録される．

-logfile *file* 通常 Julius は全てのログ出力を標準出力に出力する．このオプションを指定することで，それらの出力を指定ファイルに切替えることができる．(Rev.4.0)

-nolog ログ出力を禁止する．標準出力には何も出力されなくなる．(Rev.4.0)

-help エンジン設定，オプション一覧などのヘルプを出力して終了する．

全体オプション

全体オプションは，モデルや探索以外のオプションであり，音声入力・音検出・GMM・プラグイン・その他の設定を含む．全体オプションは，音響モデル (-AM)・言語モデル (-LM)・デコーダ (-SR) などのセクション定義の前に定義するか，-GLOBAL のあとに指定する．

オーディオ入力

-input {mic|rawfile|mfcfile|adinnet|stdin|netaudio|esd|alsa|oss} 音声入力ソースを選択する．音声波形ファイルの場合は file あるいは rawfile を指定する．HTK 形式の特徴量ファイルを認識する場合は htkparam あるいは mfcfile を指定する．起動後にプロンプトが表れるので，それに対してファイル名を入力する．adinnet では，adintool などのクライアントプロセスから音声データをネットワーク経由で受け取ることができる．netaudio は DatLink のサーバから，stdin は標準入力からの音声入力を認識する．esd は，音声デバイスの共有手段として多くの Linux のデスクトップ環境で利用されている EsounD daemon からの入力を認識する．

-filelist *filename* (-input rawfile|mfcfile 時) *filename* 内に列挙されている全てのファイルについて認識を順次行う．*filename* には認識する入力ファイル名を 1 行に 1 つずつ記述する．

-notypecheck 入力の特徴量ベクトルの型チェックを無効にする．通常 Julius は入力の型が音響モデルとマッチするかどうかをチェックし，マッチしないとエラー終了する．このオプションはそのチェックを回避する．なんらかの理由で型チェックがうまく動作しないときに使用する．

-48 48kHz で入力を行い，16kHz にダウンサンプリングしながら認識する．これは 16kHz のモデルを使用しているときのみ有効である．ダウンダンプリングの内部機能は sptk から移植された．(Rev. 4.0)

-NA *devicename* DatLink サーバのデバイス名 (-input netaudio).

-adport *port_number* -input adinnet 使用時，接続を受け付ける adinnet のボート番号を指定する．(default: 5530)

-nostrip 音声取り込み時，デバイスやファイルによっては，音声波形中に振幅が "0" となるフレームが存在することがある．Julius は通常，音声入力に含まれるそのようなフレームを除去する．この零サンプル除去がうまく動かない場合，このオプションを指定することで自動消去を無効化することができる．

-zmean , -nozmean 入力音声ストリームに対して直流成分除去を行う．全ての音声処理のの前段として処理される．-zmeansource オプションも見よ．

レベルと零交差による入力検知

-cutsilence , -nocutsilence レベルと零交差による入力検知を行うかどうかを指定する．デフォルトは，リアルタイム認識（デバイス直接入力およびネットワーク入力）では on, ファイル入力では off である．このオプションを指定することで，例えば長時間録音された音声ファイルに対して音声区間検出を行いながら認識を行うこともできる．

-lv *thres* 振幅レベルのしきい値．値は 0 から 32767 の範囲で指定する．(default: 2000)

-zc *thres* 零交差数のしきい値．値は 1 秒あたりの交差数で指定する．(default: 60)

-headmargin *msec* 音声区間開始部のマージン．単位はミリ秒．(default: 300)

-tailmargin *msec* 音声区間終了部のマージン．単位はミリ秒．(default: 400)

入力棄却 入力長，あるいは平均パワーによる入力の事後棄却が行える．平均パワーによる棄却は，デフォルトでは無効化されており，ソースからコンパイルする際に `configure` に`--enable-power-reject` を指定することで有効となる．リアルタイム認識時で，かつ特徴量でパワー項を持つ場合のみ使用できる．

-rejectshort *msec* 検出された区間長が *msec* 以下の入力を棄却する．その区間の認識は中断・破棄される．

-powerthres *thres* 切り出し区間の平均パワーのしきい値．(Rev.4.0)

このオプションはコンパイル時に`--enable-power-reject` が指定されたときに有効となる．

GMM / GMM-VAD

-gmm *hmmdefs_file* GMM 定義ファイル．3 状態（出力状態が 1 つのみ）の HMM として定義する．形式は HTK 形式で与える．形式や使用できる特徴量の制限は音響モデルと同じである．なお，GMM で用いる MFCC 特徴量の設定は，`-AM_GMM` のあとに音響モデルと同様に指定する．この特徴量設定は音響モデルと別に，明示的に指定する必要があることに注意が必要である．

-gmmnum *number* GMM 指定時，計算するガウス分布数を指定する．フレームごとに GMM の出力確率を求める際，各モデルで定義されている混合ガウス分布のうち，この *number* で指定した数の上位ガウス分布の確率のみを計算する．小さな値を指定するほど GMM の計算量を削減できるが，計算精度が悪くなる．(default: 10)

-gmmreject *string* GMM で定義されているモデル名のうち，非音声として棄却すべきモデルの名称を指定する．モデル名を複数指定することができる．複数指定する場合は，空白を入れずコンマで区切って一つの *string* として指定する．

-gmmmargin *frames* (GMM_VAD) GMM VAD による区間検出の開始部マージン．単位はフレーム数で指定する．(default: 20) (Rev. 4.0)

このオプションは`--enable-gmm-vad` 付きでコンパイルされたときに有効となる．

-gmmup *value* (GMM_VAD) 音声区間の開始とみなす VAD スコアの閾値．VAD スコアは (音声 GMM の最大尤度 - 非音声 HMM の最大尤度) で表される．(Default: 0.7) (Rev.4.1)

このオプションは`--enable-gmm-vad` 付きでコンパイルされたときに有効となる．

-gmmdown *value* (GMM_VAD) 音声区間の終了とみなす VAD スコアの閾値．VAD スコアは (音声 GMM の最大尤度 - 非音声 HMM の最大尤度) で表される．(Default: -0.2) (Rev.4.1)

このオプションは`--enable-gmm-vad` 付きでコンパイルされたときに有効となる．

デコーディングオプション デコーディングオプションは，使用する認識アルゴリズムに関する設定を行うオプションである．この設定はエンジン全体に対する設定であり，全ての認識処理インスタンスで共通の設定となる．探索の幅や言語重みなどの個々のデコーディング設定については，認識処理インスタンスごとに指定する．

-realtime , -norealtime 入力と並行してリアルタイム認識を行うかどうかを明示的に指定する．デフォルトの設定は入力デバイスに依存し，マイクロフォン等のデバイス直接認識，ネットワーク入力，および DatLink/NetAudio 入力の場合は ON, ファイル入力や特徴量入力については OFF となっている．

その他

-C ***jconffile*** jconf 設定ファイルを読み込む．ファイルの内容がこの場所に展開される．

-version バージョン情報を標準エラー出力に出力して終了する．

-setting エンジン設定情報を標準エラー出力に出力して終了する．

-quiet 出力を抑制する．認識結果は単語列のみが出力される．

-debug (デバッグ用) モデルの詳細や探索過程の記録など，様々なデバッグ情報をログに出力する．

-check {wchmm|trellis|triphone} デバッグ用のチェックモードに入る．

-plugindir ***dirlist*** プラグインを読み込むディレクトリを指定する．複数の場合はコロンで区切って並べて指定する．

複数モデル認識のためのインスタンス宣言

-AM ***name*** 音響モデルインスタンスを新たに宣言する．以降の音響モデルに関する設定はこのインスタンスに対するものと解釈される．*name* にはインスタンスにつける名前を指定する（既にある音響モデルインスタンスと同じ名前であってはいけない）. (Rev.4.0)

-LM ***name*** 言語モデルインスタンスを新たに宣言する．以降の言語モデルに関する設定はこのインスタンスに対するものと解釈される．*name* にはインスタンスにつける名前を指定する（既にある言語モデルインスタンスと同じ名前であってはいけない）. (Rev.4.0)

-SR ***name am_name lm_name*** 認識処理インスタンスを新たに宣言する．以降の認識処理や探索に関する設定はこのインスタンスに対するものと解釈される．*name* にはインスタンスにつける名前を指定する（既にある認識処理インスタンスと同じ名前であってはいけない）．*am_name*, *lm_name* にはそれぞれこのインスタンスが使用する音響モデルと言語モデルのインスタンスを名前，あるいは ID 番号で指定する．(Rev.4.0)

-AM_GMM GMM 使用時に，GMM 計算のための特徴量抽出パラメータを，この宣言のあとに指定する．もし GMM 使用時にこのオプションで GMM 用の特徴量パラメータを指定しなかった場合，最後に指定した音響モデル用の特徴量がそのまま用いられる．(Rev.4.0)

-GLOBAL 全体オプション用のセクションを開始する．-AM, -LM, -SR などのインスタンス宣言を用いる場合，音声入力設定などの全体オプションは，これらの全てのインスタンス定義よりも前か，あるいはこのオプションのあとに指定する必要がある．この全体オプション用のセクションは，jconf 内で何回現れても良い．(Rev.4.1)

-nosectioncheck , -sectioncheck 複数インスタンスを用いる jconf において，オプションの位置チェックの有効・無効を指定する．有効である場合，ある種類のインスタンスの宣言がされたあとは，他のインスタンス宣言が現れるまで，そのインスタンスのオプションしか指定できない（例：-AM のあと，他の -AM や-LM などが現れるまで，音響モデルオプションしか指定できない．他のオプションがあらわれた場合はエラーとなる）．また，全体オプションは全てのモデルインスタンスの前に指定する必要がある．デフォルトでは有効になっている．(Rev.4.1)

言語モデル (-LM)

このグループには，各モデルタイプごとに指定するオプションが含まれている．一つのインスタンスには一つのモデルタイプだけが指定可能である．

N-gram

-d ***bingram_file*** 使用する N-gram をバイナリファイル形式で指定する．バイナリ形式への変換は mkbingram を使用する．

-nlr ***arpa_ngram_file*** 前向き (left-to-right) の N-gram 言語モデルを指定する．*arpa_ngram_file* は ARPA 標準形式のファイルである必要がある．

-nrl ***arpa_ngram_file*** 後ろ向き (right-to-left) の N-gram 言語モデルを指定する．*arpa_ngram_file* は ARPA 標準形式のファイルである必要がある．

-v *dict_file* N-gram，または文法用の単語辞書ファイルを指定する．

-silhead *word_string* -siltail *word_string* 音声入力両端の無音区間に相当する「無音単語」エントリを指定する．単語の読み（N-gram エントリ名），あるいは"#"+単語番号（辞書ファイルの行番号-1）で指定する．デフォルトはそれぞれ "<s>", "</s>" である．

-mapunk *word_string* unknown に対応する単語名を指定する．デフォルトは，"<unk>" あるいは "<UNK>" である．この単語は，認識辞書において N-gram にない単語を指定した場合にマッピングされる単語である．

-iwspword ポーズに対応する無音単語を辞書に追加する．追加される単語の内容はオプション`-iwspentry`で変更できる．

-iwspentry *word_entry_string* `-iwspword`指定時に追加される単語エントリの内容を指定する．辞書エントリと同じ形式で指定する．(default: "<UNK> [sp] sp sp")

-sepnum *number* 木構造化辞書の構築時に線形登録する単語数を指定する．(default: 150)

文法 `-gram`や`-gramlist`で文法を複数回指定することで，一つのインスタンス内で複数の文法を用いることができる．(旧 Julius のオプション `-dfa`, `-v` の組合せは単一の文法のみ指定可能である）

-gram gramprefix1[,gramprefix2[,gramprefix3,...]] 認識に使用する文法を指定する．文法はファイル（辞書および構文制約有限オートマトン）のプレフィックスで指定する．すなわち，ある認識用文法が `dir/foo.dict` と `dir/foo.dfa` としてあるとき，`dir/foo` のように拡張子を省いた名前で指定する．文法はコンマで区切って複数指定することができる．また繰り返し使用することでも複数指定できる．

-gramlist *list_file* 認識に使用する文法のリストをファイルで指定する．*list_file* には，`-gram` と同様の文法プレフィックスを 1 行に 1 つずつ記述する．また，このオプションを繰り返し使用することで，複数のリストファイルを指定できる．なお，リスト内で文法を相対パスで指定した場合，それは，そのリストファイルからの相対パスとして解釈されることに注意が必要である．

-dfa *dfa_file* -v *dict_file* 認識に使用する文法の構文制約オートマトンと辞書をそれぞれ指定する．(Julius-3.x との互換性のための古いオプションであり，使用すべきでない）

-nogram それまでに`-gram`，`-gramlist`，`-dfa` および `-v` で指定された文法のリストをクリアし，文法の指定なしの状態にする．

単単語

-w *dict_file* 単単語認識で用いる単語辞書を指定する．ファイル形式は単語 N-gram や文法と同一である．辞書上の全ての単語が認識対象となる．(Rev.4.0)

-wlist *list_file* 単語辞書のリストを指定する．*list_file* には 1 行に一つずつ，使用する単語辞書のパスを記述する．相対パスを用いた場合，それはその *list_file* からの相対パスとして解釈される． (Rev.4.0)

-nogram それまでに`-w` あるいは `-wlist` で指定された辞書のリストをクリアし，指定なしの状態に戻す．

-wsil *head_sil_model_name tail_sil_model_name sil_context_name* 単単語認識時，音声入力の両端の無音モデルおよびそのコンテキスト名を指定する．*sil_context_name* として NULL を指定した場合，各モデル名がそのままコンテキストとして用いられる．

User-defined LM

-userlm プログラム中のユーザ定義言語スコア計算関数を使用することを宣言する．(Rev.4.0)

その他の言語モデル関連

-forcedict 単語辞書読み込み時のエラーを無視する．通常 Julius は単語辞書内にエラーがあった場合そこで動作を停止するが，このオプションを指定することで，エラーの生じる単語をスキップして処理を続行することができる．

音響モデル・特徴量抽出 (-AM) (-AM_GMM)

音響モデルオプションは，音響モデルおよび特徴量抽出・フロントエンド処理に関する設定を行う．特徴量抽出，正規化処理，スペクトルサブトラクションの指定もここで行う．

音響 HMM 関連

-h *hmmdef_file* 使用する HMM 定義ファイル．HTK の ASCII 形式ファイル，あるいは Julius バイナリ形式のファイルのどちらかを指定する．バイナリ形式へは mkbinhmm で変換できる．

-hlist *hmmlist_file* HMMlist ファイルを指定する．テキスト形式，あるいはバイナリ形式のどちらかを指定する．バイナリ形式へは mkbinhmmlist で変換できる．

-tmix *number* Gaussianpruning の計算状態数を指定する．小さいほど計算が速くなるが，音響尤度の誤差が大きくなる．See also -gprune. (default: 2)

-spmodel *name* 文中のショートポーズに対応する音韻 HMM の名前を指定する．このポーズモデル名は，-iwsp, -spsegment, -pausemodels に関係する．また，文法使用時にスキップ可能なポーズ単語エントリの識別にも用いられる．(default: "sp")

-multipath -nomultipath 状態間遷移を拡張するマルチパスモードを有効にする．オプション指定がない場合，Julius は音響モデルの遷移をチェックし，必要であれば自動的にマルチパスモードを有効にする．このオプションは，ユーザが明示的にモードを指定したい場合に使用する．

この機能は 3.x ではコンパイル時オプションであったが，4.0 より実行時オプションとなった．(rev.4.0)

-gprune {safe|heuristic|beam|none|default} 使用する Gaussian pruning アルゴリズムを選択する．none を指定すると Gaussian pruning を無効化しすべてのガウス分布について厳密に計算する．safe は上位 N 個を計算する．heuristic と beam は safe に比べてより積極的な枝刈りを行うため計算量削減の効果が大きいが，認識精度の低下を招く可能性がある．default が指定された場合，デフォルトの手法を使う．(default: tied-mixture model の場合，standard 設定では safe，fast 設定では beam．tied-mixture でない場合 none).

-iwcd1 {max|avg|best number} 第 1 パスの単語間トライフォン計算法を指定する．max 指定時，同じコンテキストのトライフォン集合の全尤度の最大値を近似尤度として用いる．avg は全尤度の平均値を用いる．best number は上位 N 個のトライフォンの平均値を用いる．デフォルトは，一緒に使用される言語モデルに依存する．N-gram 使用時には best 3，文法使用時は avg となる．もしこの音響モデルが異なるタイプの複数の言語モデルで共有される場合は，後に定義されたほうのデフォルトがデフォルト値として用いられる．

-iwsppenalty *float* -iwsp によって末尾に付加される単語末ショートポーズの挿入ペナルティ．ここで指定した値が，通常単語の末尾から単語末ショートポーズへの遷移に追加される．

-gshmm *hmmdef_file* Gaussian Mixture Selection 用のモノフォン音響モデルを指定する．GMS 用モノフォンは通常のモノフォンから mkgshmm によって生成できる．

-gsnum *number* GMS 使用時，対応するトライフォンを詳細計算するモノフォンの状態の数を指定する． (default: 24)

特徴量抽出パラメータ

-smpPeriod *period* 音声のサンプリング周期を指定する．単位は，100 ナノ秒の単位で指定する．サンプリング周期は -smpFreq でも指定可能．(default: 625)

このオプションは HTK の SOURCERATE に対応する．同じ値が指定できる．

複数の音響モデルを用いる場合，全インスタンスで共通の値を指定する必要がある．

-smpFreq *Hz* 音声のサンプリング周波数 (Hz) を指定する．(default: 16,000)

複数の音響モデルを用いる場合，全インスタンスで共通の値を指定する必要がある．

-fsize *sample_num* 窓サイズをサンプル数で指定 (default: 400).

このオプションは HTK の WINDOWSIZE に対応する．ただし値は HTK と異なり，(HTK の値 / smpPeriod) となる．

複数の音響モデルを用いる場合，全インスタンスで共通の値を指定する必要がある．

-fshift ***sample_num*** フレームシフト幅をサンプル数で指定 (default: 160).

このオプションは HTK の TARGETRATE に対応する．ただし値は HTK と異なり，(HTK の値 / smpPeriod) となる．

複数の音響モデルを用いる場合，全インスタンスで共通の値を指定する必要がある．

-preemph ***float*** プリエンファシス係数 (default: 0.97)

このオプションは HTK の PREEMCOEF に対応する．同じ値が指定できる．

-fbank ***num*** フィルタバンクチャンネル数．(default: 24)

このオプションは HTK の NUMCHANS に対応する．同じ値が指定できる．指定しないときのデフォルト値が HTK と異なっていることに注意（HTK では 22）．

-ceplif ***num*** ケプストラムのリフタリング係数. (default: 22)

このオプションは HTK の CEPLIFTER に対応する．同じ値が指定できる．

-rawe ，**-norawe** エネルギー項の値として，プリエンファシス前の raw energy を使用する / しない (default: disabled=使用しない)

このオプションは HTK の RAWENERGY に対応する．指定しないときのデフォルトが HTK と異なっていることに注意（HTK では enabled).

-enormal ，**-noenormal** エネルギー項の値として，発話全体の平均で正規化した正規化エネルギーを用いるかどうかを指定する．(default: -noenormal)

このオプションは HTK の ENORMALISE に対応する．指定しないときのデフォルトが HTK と異なっていることに注意（HTK では enabled).

-escale ***float_scale*** エネルギー正規化時の，対数エネルギー項のスケーリング係数．(default: 1.0)

このオプションは HTK の ESCALE に対応する．デフォルト値が HTK と異なっていることに注意（HTK では 0.1）．

-silfloor ***float*** エネルギー正規化時の，無音部のエネルギーのフロアリング値．(default: 50.0)

このオプションは HTK の SILFLOOR に対応する．同じ値が指定できる．

-delwin ***frame*** 一次差分計算用のウィンドウフレーム幅．(default: 2)

このオプションは HTK の DELTAWINDOW に対応する．同じ値が指定できる．

-accwin ***frame*** 二次差分計算用のウィンドウフレーム幅．(default: 2)

このオプションは HTK の ACCWINDOW に対応する．同じ値が指定できる．

-hifreq ***Hz*** MFCC のフィルタバンク計算時におけるバンド制限を有効化する．このオプションではカットオフ周波数の上限値を指定する．-1 を指定することで無効化できる．(default: -1)

このオプションは HTK の HIFREQ に対応する．同じ値が指定できる．

-lofreq ***Hz*** MFCC のフィルタバンク計算時におけるバンド制限を有効化する．このオプションではカットオフ周波数の下限値を指定する．-1 を指定することで無効化できる．(default: -1)

このオプションは HTK の LOFREQ に対応する．同じ値が指定できる．

-zmeanframe ，**-nozmeanframe** 窓単位の直流成分除去を有効化／無効化する．(default: disabled)

このオプションは HTK の ZMEANSOURCE に対応する．-zmean も参照のこと．

-usepower フィルタバンク解析で振幅の代わりにパワーを使う．(default: disabled)

正規化処理

-cvn ケプストラム分散正規化 (cepstral variance normalization; CVN) を有効にする．ファイル入力では，入力全体の分散に基づいて正規化が行われる．直接入力ではあらかじめ分散が得られないため，最後の入力の分散で代用される．音声信号入力でのみ有効である．

-vtln ***alpha lowcut hicut*** 周波数ワーピングを行う．声道長正規化 (vocal tract length normalization; VTLN) に使用できる．引数はそれぞれワーピング係数，周波数上端，周波数下端であり，HTK 設定の WARPFREQ，WARPHCUTOFF および WARPLCUTOFF に対応する．

-cmnload ***file*** 起動時にケプストラム平均ベクトルを *file* から読み込む．ファイルは-cmnsave で保存されたファイルを指定する．これは MAP-CMN において，起動後最初の発話においてケプストラム平均の初期値として用いられる．通常，2 発話目以降は初期値は，直前の入力の平均に更新されるが，-cmnnoupdate を指定された場合，常にこのファイルの値が各発話の初期値として用いられる．

-cmnsave ***file*** 認識中に計算したケプストラム平均ベクトルを *file* へ保存する．すでにファイルがある場合は上書きされる．この保存は音声入力が行われるたびに上書きで行われる．

-cmnupdate -cmnnoupdate 実時間認識時，初期ケプストラム平均を入力ごとに更新するかどうかを指定する．通常は有効 (-cmnupdate) であり，過去 5 秒間の入力の平均を初期値として更新する．-cmnnoupdate が指定された場合，更新は行われず，初期値は起動時の値に固定される．-cmnload で初期値を指定することで，常に同じ初期値を使うようにすることができる．

-cmnmapweight ***float*** MAP-CMN の初期ケプストラム平均への重みを指定する．値が大きいほど初期値に長時間依存し，小さいほど早く現入力のケプストラム平均を用いるようになる．(default: 100.0)

フロントエンド処理

-sscalc 入力先頭の無音部を用いて，入力全体に対してスペクトルサブトラクションを行う．先頭部の長さは-sscalclen で指定する．ファイル入力に対してのみ有効である．-ssload と同時に指定できない．

-sscalclen ***msec*** -sscalc オプション指定時，各ファイルにおいてノイズスペクトルの推定に用いる長さをミリ秒で指定する．(default: 300)

-ssload ***file*** ノイズスペクトルを *file* から読み込んでスペクトルサブトラクションを行う．*file* はあらかじめ mkss で作成する．マイク入力・ネットワーク入力などのオンライン入力でも適用できる．-sscalc と同時に指定できない．

-ssalpha ***float*** -sscalc および-ssload 用の減算係数を指定する．この値が大きいほど強くスペクトル減算を行うが，減算後の信号の歪も大きくなる．(default: 2.0)

-ssfloor ***float*** スペクトルサブトラクションのフロアリング係数を指定する．スペクトル減算時，計算の結果パワースペクトルが負となってしまう帯域に対しては，原信号にこの係数を乗じたスペクトルが割り当てられる．(default: 0.5)

その他の音響モデル関連オプション

-htkconf ***file*** HTK Config ファイルを解析して，対応する特徴量抽出オプションを Julius に自動設定する．*file* は HTK で音響モデル学習時に使用した Config ファイルを指定する．なお，Julius と HTK ではパラメータのデフォルト値が一部異なるが，このオプションを使用する場合，デフォルト値も HTK のデフォルトに切替えれられる．

認識処理・探索 (-SR)

認識処理・探索オプションは，第 1 パス・第 2 パス用のビーム幅や言語重みのパラメータ，ショートポーズセグメンテーションの設定，単語ラティス・CN 出力用設定，forced alignment の指定，その他の認識処理と結果出力に関するパラメータを含む．

第 1 パスパラメータ

-lmp ***weight penalty*** (N-gram 使用時) 第 1 パス用の言語スコア重みおよび挿入ペナルティ．ペナルティは負であれば単語挿入を抑制し，正であれば単語挿入を促進する．

-penalty1 ***penalty*** (文法使用時) 第 1 パス用の単語挿入ペナルティ． (default: 0.0)

-b ***width*** 第 1 パス探索の枝刈り (rank pruning) のビーム幅を指定する．単位は HMM ノード数である．デフォルト値は音響モデルやエンジンの設定による．モノフォン使用時は 400, トライフォン使用時は 800，トライフォンでかつ setup=v2.1 のときは 1000 となる．

-nlimit ***num*** 第 1 パスでノードごとに保持する仮説トークンの最大数．通常は 1 で固定されており変更できない．コンパイル時に `configure` で`--enable-wpair` および`--enable-wpair-nlimit` が指定されているときのみ変更できる．

-progout 第 1 パスで，一定時間おきにその時点での最尤仮説系列を出力する．

-proginterval ***msec*** `-progout` の出力インターバルをミリ秒で指定する．(default: 300)

第 2 パスパラメータ

-lmp2 ***weight penalty*** (N-gram 使用時) 第 2 パス用の言語スコア重みおよび挿入ペナルティ．ペナルティは負であれば単語挿入を抑制し，正であれば単語挿入を促進する．

-penalty2 ***penalty*** (文法使用時) 第 2 パス用の単語挿入ペナルティ． (default: 0.0)

-b2 ***width*** 第 2 パス探索における仮説展開回数の上限を指定する．単位は仮説数．(default: 30)

-sb ***float*** 第 2 パスの仮説尤度計算時のスコア幅を指定する．単位は対数尤度差である．(default: 80.0)

-s ***num*** 仮説のスタックサイズを指定する．(default: 500)

-n ***num*** *num* 個の文仮説数が見付かるまで探索を行う．得られた仮説はスコアでソートされて出力される（`-output` も見よ）．デフォルト値はコンパイル時のエンジン設定によって変わり，fast 版では 1, standard 版では 10 である．

-output ***num*** 見つかった N-best 候補のうち，結果として出力する文仮説の数を指定する．`-n` も参照のこと．(default: 1)

-m ***count*** 探索打ち切りのための仮説展開回数のしきい値を指定する．(default: 2000)

-lookuprange ***frame*** 第 2 パスの単語展開時に，接続しうる次単語候補を見付けるための終端時刻の許容幅をフレーム数で指定する．値を大きくするほどその周辺の多くの仮説を次単語候補として仮説展開が行われるようになるが，探索が前に進みにくくなることがある．(default: 5)

-looktrellis 仮説展開を第 1 パスの結果単語トレリス上に絞る．

ショートポーズセグメンテーション

-spsegment ショートポーズセグメンテーションを有効にする． (Rev.4.0)

-spdur ***frame*** 無音区間判定のためのしきい値を指定する．無音単語が一位仮説となるフレームがこの値以上続いたとき，無音区間として入力が区切られる．(default: 10)

-pausemodels ***string*** 「無音単語」を定義するための音響モデルにおける無音モデルの名前を指定する．コンマで区切って複数の名前を指定できる．このオプションが指定されない場合，文法を用いた認識では`-spmodel` で指定されるモデルのみを読みとする単語が無音単語とされる．また，N-gram ではこれに加えて`-silhead` および `-siltail` で指定される単語も無音単語として扱われる．(Rev.4.0)

-spmargin ***frame*** デコーダベース VAD において，アップトリガ時の巻戻し幅をフレーム数で指定する．(Rev.4.0)

このオプションは `configure` に`--enable-decoder-vad` を付けてコンパイルしたときのみ有効である．

-spdelay ***frame*** デコーダベース VAD において，アップトリガ判定の遅延幅をフレーム数で指定する．(Rev.4.0)

このオプションは `configure` に`--enable-decoder-vad` を付けてコンパイルしたときのみ有効である．

単語ラティス / confusion network 出力

-lattice , -nolattice 単語グラフ（ラティス）の出力を有効化/無効化する．

-confnet , -noconfnet Confusion network の出力を有効化/無効化する．confusion network は単語グラフから生成されるため，有効時は同時に-lattice も有効化される．(Rev.4.0)

-graphrange *frame* グラフ生成において近傍の同一単語仮説をマージする．開始フレームおよび終了フレームの位置の差がそれぞれ *frame* 以下の同一単語仮説についてマージする．その際，スコアは高いほうのものが残される．値が -1 の場合，マージは一切行われない．値を大きくするほどコンパクトなグラフが生成されるが，スコアの誤差が大きくなる．このオプションは-confnet にも影響する．(default: 0)

-graphcut *depth* 生成されたグラフに対して，深さによるカットオフを行う．*depth* は，あるフレームにおいて存在可能な単語数の上限を指定する．Julius では，第 2 パスの探索が不安定な場合，一部分が極端に深いグラフが生成されることが稀にあり，このオプションによってそれを抑制することができる．-1 を指定することでこの機能は無効化される．(default: 80)

-graphboundloop *count* 事後的に行われる単語グラフの境界時間調整において，振動による無限ループを防ぐための打ち切り値を指定する．(default: 20)

-graphsearchdelay , -nographsearchdelay 巨大グラフ生成用にアルゴリズムをチューニングする．このオプションが有効時，Julius は第 1 文仮説が見つかる前のグラフ生成時の仮説中断を行わないように，グラフ生成アルゴリズムを変更する．これは，ビーム幅や探索範囲を極端に大きくして巨大なワードグラフを生成しようとするときに，グラフの精度を改善することがある．(default: disabled)

複数文法/複数辞書認識 文法や単単語認識において，一つのインスタンスで複数の文法や辞書を用いる場合に指定できるオプションである．

-multigramout , -nomultigramout 複数文法あるいは複数辞書を用いて認識を行う場合，通常の Julius は全ての文法/辞書の中から最尤仮説を出力する．このオプションを指定することで，与えられた個々の文法/辞書ごとに一位仮説を出力することができる．(default: disabled)

Forced alignment

-walign 認識結果を用いて，入力に対する単語単位の forced alignment を行う．単語の境界フレームと平均音響尤度が出力される．

-palign 認識結果を用いて，入力に対する音素単位の forced alignment を行う．音素ごとの境界フレームと平均音響尤度が出力される．

-salign 認識結果を用いて，入力に対する HMM の状態単位の forced alignment を行う．状態ごとの境界フレームと平均音響尤度が出力される．

その他

-inactive 認識処理インスタンスを一時停止状態 (inactive state) で起動する．(Rev.4.0)

-1pass 第 1 パスのみを実行する．このオプションを指定した場合，第 2 パスは実行されない．

-fallback1pass 通常，第 2 パスの探索が失敗したとき，Julius は認識結果無しで終了する．このオプションを指定することで，そのような第 2 パスの失敗時に，第 1 パスの最尤仮説を最終結果として出力することができる．（これは Julius-3.x でのデフォルトの振る舞いである）

-no_ccd , -force_ccd 音響モデルを音素コンテキスト依存モデルとして扱うかどうかを明示的に指定する．デフォルトは HMM 中のモデル名から自動判断される．

-cmalpha *float* 確信度計算のためのスコアのスムージング係数．(default: 0.05)

-iwsp （マルチパスモード時のみ有効）単語間にショートポーズモデルを挟み込んだ認識処理を行う．このオプションを指定すると，辞書上の全単語の末尾に，スキップ可能なショートポーズモデルが付加される．このショートポーズモデルはコンテキストを考慮せず，また前後の音素のコンテキストにも表れない．付加するショートポーズモデルは-spmodel で指定できる．

-transp *float* 透過単語に対する追加の挿入ペナルティを指定する．(default: 0.0)

-demo -progout -quiet と同等．

ENVIRONMENT VARIABLES

ALSADEV (マイク入力で alsa デバイス使用時) 録音デバイス名を指定する．指定がない場合は "default".

AUDIODEV (マイク入力で oss デバイス使用時) 録音デバイス名を指定する．指定がない場合は "/dev/dsp".

PORTAUDIO_DEV (portaudio V19 使用時) 録音デバイス名を指定する．具体的な指定方法は adinrec の初期化時にログに出力されるので参照のこと．

LATENCY_MSEC Linux (alsa/oss) および Windows で，マイク入力時の遅延時間をミリ秒単位で指定する．短い値を設定することで入力遅延を小さくできるが，CPU の負荷が大きくなり，また環境によってはプロセスや OS の挙動が不安定になることがある．最適な値は OS やデバイスに大きく依存する．デフォルト値は動作環境に依存する．

EXAMPLES

使用例については付属のチュートリアルをご覧下さい．

SEE ALSO

julian(1), jcontrol(1), adinrec(1), adintool(1), mkbingram(1), mkbinhmm(1), mkgsmm(1), wav2mfcc(1), mkss(1)
http://julius.sourceforge.jp/

DIAGNOSTICS

正常終了した場合，Julius は exit status として 0 を返します．エラーが見付かった場合は異常終了し，exist status として 1 を返します．入力ファイルが見つからない場合やうまく読み込めなかった場合は，そのファイルに対する処理をスキップします．

BUGS

使用できるモデルにはサイズやタイプに若干の制限があります．詳しくはパッケージに付属のドキュメントを参照してください．バグ報告・問い合わせ・コメントなどは julius-info at lists.sourceforge.jp までお願いします．

COPYRIGHT

AUTHORS

Rev.1.0 (1998/02/20) 設計：河原達也と李 晃伸 (京都大学)
実装：李 晃伸 (京都大学)

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/
実装：李 晃伸 (京都大学)

Rev.3.2 (2001/08/15), Rev.3.3 (2002/09/11), Rev.3.4 (2003/10/01), Rev.3.4.1 (2004/02/25), Rev.3.4.2 (2004/04/30)
実装：李 晃伸 (奈良先端科学技術大学院大学)

Rev.3.5 (2005/11/11), Rev.3.5.1 (2006/03/31), Rev.3.5.2 (2006/07/31), Rev.3.5.3 (2006/12/29), Rev.4.0 (2007/12/19), Rev.4.1 (
実装：李 晃伸 (名古屋工業大学)

THANKS TO

このプログラムは Rev.3.1 まで，情報処理振興事業協会 (IPA) 独創的情報技術育成事業「日本語ディクテーションの基本ソフトウェアの開発」(代表者：鹿野清宏 奈良先端科学技術大学院大学教授) の援助を受けて行われました．Rev.3.4.2 までは「情報処理学会 連続音声認識コンソーシアム」において公開されました．

3.x 時代のマルチプラットフォーム DLL 版 は，板野秀樹氏 (現名城大学) の手によって作成・公開されました．また，Windows Microsoft Speech API 対応版は住吉貴志氏 (京都大学・当時) の手によるものです．

そのほか，上記の協力・貢献してくださった方々，およびさまざまな助言・コメントをいただく関係者各位に深く感謝いたします．

C.2 jcontrol

名前

jcontrol – Julius モジュールモード用のサンプルクライアント

Synopsis

`jcontrol` *`hostname`* [*`portnum`*]

Description

jcontrol は，モジュールモードで動作している julius に接続し，API を介してコントロールする簡単なコンソールプログラムです．Julius への一時停止や再開などのコマンドの送信，および Julius からの認識結果や音声イベントのメッセージ受信を行うことができます．

起動後，jcontrol は，指定ホスト上において「モジュールモード」で動作中の Julius に対し，接続を試みます．接続確立後，jcontrol はユーザーからのコマンド入力およびメッセージ受信待ち状態となります．

jcontrol は ユーザーが入力したコマンドを解釈し，対応する API コマンドを Julius へ送信します．また，Julius から認識結果や入力トリガ情報 などのメッセージが送信されてきたときは，その内容を標準出力へ書き出します．

モジュールモードの仕様については，関連文書をご覧下さい．

Options

`hostname` 接続先のホスト名

`portnum` ポート番号（デフォルト：10500）

COMMANDS

jcontrol は標準入力から 1 行ずつコマンド文字列を受け取る．コマンドの一覧は以下の通り．

動作制御

`pause` Julius の認識動作を中断させ，一時停止状態に移行させる．一時停止状態にある Julius は，たとえ音声入力があっても認識処理を行わない．ある区間の音声認識処理の途中でこのコマンドを受け取った場合，Julius はその認識処理が終了した後，一時停止状態に移行する．

`terminate` `pause` と同じく，Julius の認識動作を中断させ，一時停止状態に移行させる．ある区間の音声認識処理の途中でこのコマンドを受け取った場合，その入力を破棄して即座に一時停止状態に移行する．

`resume` Julius を一時停止状態から通常状態へ移行させ，認識を再開させる．

`inputparam` ***`arg`*** 文法切り替え時に音声入力であった場合の入力中音声の扱いを指定．`"TERMINATE"`, `"PAUSE"`, `"WAIT"`のうちいずれかを指定．

`version` Julius にバージョン文字列を返させる．

`status` Julius からシステムの状態 (active / sleep) を報告させる．

文法・単語認識関連

graminfo カレントプロセスが保持している文法の一覧をクライアントへ出力させる．

changegram *prefix* カレントプロセスの認識文法を "prefix.dfa" と"prefix.dict" に入れ替える．カレントプロセス内の文法は全て消去され，指定された文法に置き換わる．

カレントプロセスが孤立単語認識の場合，"*prefix*" の変わりに辞書ファイルのみを"filename.dict" の形で指定する．

addgram *prefix* 認識文法として "prefix.dfa" と"prefix.dict" をカレントプロセスに追加する．

カレントプロセスが孤立単語認識の場合，"*prefix*" の変わりに辞書ファイルのみを"filename.dict" の形で指定する．

deletegram *gramlist* カレントプロセスから指定された文法を削除する．文法の指定は，文法名（追加時の *prefix*）か，あるいは Julius から送られる GRAMINFO 内にある文法 ID で指定する．複数の文法を削除したい場合は，文法名もしくは ID をカンマで区切って複数指定する（ID と文法名が混在してもよい）．

deactivategram *gramlist* カレントプロセスの指定された文法を一時的に無効にする．無効にされた文法は，エンジン内に保持されたまま，認識処理からは一時的に除外される．無効化された文法は activategram で再び有効化できる．

文法の指定は，文法名（追加時の *prefix*）か，あるいは Julius から送られる GRAMINFO 内にある文法 ID で指定する．複数の文法を指定したい場合は，文法名もしくは ID をカンマで区切って複数指定する（ID と文法名が混在してもよい）．

activategram *gramlist* カレントプロセスで無効化されている文法を有効化する．文法の指定は，文法名（追加時の *prefix*）か，あるいは Julius から送られる GRAMINFO 内にある文法 ID で指定する．複数の文法を指定したい場合は，文法名もしくは ID をカンマで区切って複数指定する（ID と文法名が混在してもよい）．

addword *grammar_name_or_id dictfile* dictfile の中身を，カレントプロセスの指定された文法に追加する．

syncgram addgram や deletegram などによる文法の更新を即時に行う．同期確認用である．

プロセス関連のコマンド

Julius-4 では複数モデルの同時認識が行える．この場合，認識プロセス ("-SR" で指定された認識処理インスタンス) ごとにモジュールクライアントから操作を行うことができる．

クライアントからはどれか一つのプロセスが「カレントプロセス」として割り当てられる．文法関連の命令はカレントプロセスに対して行われる．

listprocess Julius に現在エンジンにある認識プロセスの一覧を送信させる．

currentprocess *procname* カレントプロセスを指定された名前のプロセスに切り替える．

shiftprocess カレントプロセスを循環切り替えする．呼ばれるたびにその次のプロセスにカレントプロセスが切り替わる．

addprocess *jconffile* エンジンに認識プロセスを新たに追加する．与える *jconffile* は，通常のものと違い，ただ一種類の LM 設定を含むものである必要がある．また，実際に送られるのはパス名のみであり，ファイル読み込みは Julius 側で行われるため，ファイルパスは Julius から見える場所を指定する必要が有る．

追加された LM および認識プロセスは，jconffile の名前がプロセス名となる．

delprocess *procname* 指定された名前の認識プロセスをエンジンから削除する．

deactivateprocess *procname* 指定された名前の認識プロセスを，一時的に無効化する．無効化されたプロセスは次回以降の入力に対して認識処理からスキップされる．無効化されたプロセスは activateprocess で再び有効化できる．

activateprocess *procname* 指定された名前の認識プロセスを有効化する．

EXAMPLES

Julius からのメッセージは "> " を行の先頭につけてそのまま標準出力に出力されます．以下は実行例です．

```
% julius -C ... -module
```

上記のようにして Julius をモジュールモードで起動した後，jcontrol をそのホスト名を指定して起動します．

```
% jcontrol hostname
```

音声入力を行えば，イベント内容や結果が jcontrol 側に送信されます．jcontrol に対してコマンドを入力する（最後に Enter を押す）と，Julius にコマンドが送信され，Julius が制御されます．
詳しいプロトコルについては，関連文書を参照してください．

SEE ALSO

julius (1)

COPYRIGHT

LICENSE

C.3 jclient.pl

名前

jclient.pl – perl 版サンプルクライアント

Synopsis

```
jclient.pl
```

Description

Julius に付属のサンプルクライアント "jcontrol" の Perl 版です．モジュール（サーバ）モードで動く Julius から認識結果を受け取ったり，Julius を制御したりできます．
わずか 57 行の簡単なプログラムです．アプリケーションへ Julius を組み込む際の参考になれば幸いです．ご自由にご利用ください。

EXAMPLES

```
% julius -C ... -module
```

上記のようにして Julius をモジュールモードで起動した後，jclient.pl を起動します．接続するホストのデフォルトは localhost, ポート番号は 10500 です．変えたい場合はスクリプトの冒頭を書き換えてください．

```
% jclient.pl
```

音声入力を行えば，イベント内容や結果が jclient.pl 側に送信され，標準出力に出力されます．また，jclient.pl に対してコマンドを入力する（最後に Enter を押す）と，Julius にコマンドが送信され，Julius が制御されます．コマンドは，仕様書にあるモジュールコマンドを生のまま記述します．

SEE ALSO

julius (1) , jcontrol (1)

COPYRIGHT

C.4 mkbingram

名前

mkbingram – バイナリ N-gram 変換

Synopsis

`mkbingram` [-nlr *`forward_ngram.arpa`*] [-nrl *`backward_ngram.arpa`*] [-d *`old_bingram_file`*] output_bingram_file

Description

mkbingram は，ARPA 形式の N-gram 定義ファイルを Julius 用のバイナリ N-gram ファイルに変換するツールです．あらかじめ変換しておくことで，Julius の起動を大幅に高速化できます．

Julius-4 より，N-gram は前向き，後ろ向き，あるいは両方を指定できるようになりました．mkbingram でも，どちらか一方だけでバイナリ N-gram を作成することができます．また，両方を指定した場合は，それら 2 つの N-gram は一つのバイナリ N-gram に結合されます．

前向き N-gram のみが指定されたとき，mkbingram は 前向き N-gram だけからバイナリ N-gram を生成します．このバイナリ N-gram を使うとき，Julius はその中の 2-gram を使って第 1 パスを行い，第 2 パスではその前向き確率から後向きの確率を，ベイズ則に従って算出しながら認識を行います．

後向き N-gram のみが指定されたとき，mkbingram は後ろ向き N-gram だけからバイナリ N-gram を生成します．このバイナリ N-gram を使うとき，Julius はその中の後向き 2-gram からベイズ則に従って算出しながら第 1 パスの認識を行い，第 2 パスでは後向き N-gram を使った認識を行います．

両方が指定されたときは，前向き N-gram 中の 2-gram と後向き N-gram が統合されたバイナリ N-gram が生成されます．Julius ではその前向き 2-gram で第 1 パスを行い，後向き N-gram で第 2 パスを行います．なお両 N-gram は同一のコーパスから同 一の条件（カットオフ値，バックオフ計算方法等）で学習されてあり，同一の語彙を持っている必要があります．

なお，mkbingram は gzip 圧縮された ARPA ファイルもそのまま読み込めます．

バージョン 3.x 以前で作成したバイナリ N-gram は，そのまま 4.0 でも読めます．mkbingram に `-d` で与えることで，古いバイナリ形式を新しいバイナリ形式に変換することもできます．なお，4.0 以降の mkbingram で作成したバイナリ N-gram ファイルは 3.x 以前のバージョンでは使えませんのでご注意ください．

Options

`-nlr` ***`forward_ngram.arpa`*** 前向き（left-to-right）の ARPA 形式 N-gram ファイルを読み込む

`-nrl` ***`backward_ngram.arpa`*** 後ろ向き（right-to-left）の ARPA 形式 N-gram ファイルを読み込む

`-d` ***`old_bingram_file`*** バイナリ N-gram を読み込む（古いバイナリ形式の変換用）

`-swap` 文頭記号 <s> と文末記号 </s> を入れ替える．

`output_bingram_file` 出力先のバイナリ N-gram ファイル名

EXAMPLES

ARPA 形式の N-gram をバイナリ形式に変換する（前向き+後ろ向き）：

```
% mkbingram -nlr 2gram.arpa -nrl rev-Ngram.arpa outfile
```

ARPA 形式の前向き 4-gram をバイナリ形式に変換する（前向きのみ）：

```
% mkbingram -nlr 4gram.arpa outfile
```

古いバイナリ N-gram ファイルを現在の形式に変換する：

```
% mkbingram -d old_bingram new_bingram
```

SEE ALSO

julius (1) , mkbinhmm (1)

COPYRIGHT

LICENSE

C.5 mkbinhmm

名前

mkbinhmm – バイナリ HMM 変換

Synopsis

mkbinhmm [-htkconf *HTKConfigFile*] hmmdefs_file binhmm_file

Description

mkbinhmm は，HTK のアスキー形式の HMM 定義ファイルを，Julius 用のバイナリ形式へ変換します．これを使うことで Julius の起動を高速化することができます.

この音響モデルの特徴抽出条件を出力ファイルのヘッダに埋め込むことができます．埋め込むには，学習時に特徴量抽出に用いた HTK Config ファイルを"-htkconf" で指定します．ヘッダに抽出条件を埋め込むことで，認識時に自動的に必要な特徴抽出パラメータがセットされるので，便利です.

入力として，HTK アスキー形式のほかに，既に変換済みの Julius 用バイナリ HMM を与えることもできます．-htkconf と併用すれば，既存のバイナリ HMM に特徴量抽出条件パラメータを埋め込むことができます.

mkbinhmm は gzip 圧縮された HMM 定義ファイルをそのまま読み込めます.

Options

-htkconf ***HTKConfigFile*** 学習時に特徴量抽出に使用した HTK Config ファイルを指定する．指定された場合，その中の設定値が出力ファイルのヘッダに埋め込まれる．入力に既にヘッダがある場合上書きされる.

hmmdefs_file 変換元の音響モデル定義ファイル (MMF)．HTK ASCII 形式，あるいは Julius バイナリ形式.

hmmdefs_file Julius 用バイナリ形式ファイルの出力先.

EXAMPLES

HTK ASCII 形式の HMM 定義をバイナリ形式に変換する：

```
% mkbinhmm hmmdefs.ascii binhmm
```

HTK の設定ファイル Config の内容をヘッダに書き込んで出力：

```
% mkbinhmm -htkconf Config hmmdefs.ascii binhmm
```

古いバイナリ形式ファイルにヘッダ情報だけ追加する：

```
% mkbingram -htkconf Config old_binhmm new_binhmm
```

SEE ALSO

julius (1) , mkbingram (1)

COPYRIGHT

LICENSE

C.6 mkbinhmmlist

名前

mkbinhmmlist – HMMList ファイルをバイナリ形式に変換

Synopsis

`mkbinhmmlist` hmmdefs_file HMMList_file output_binhmmlist_file

Description

mkbinhmmlist は，主にトライフォンとともに使用される HMMList ファイルをバイナリ形式に変換します. 通常のテキスト形式の代わりにこれを使うことで Julius の起動を高速化することができます.

変換には，HMMList ファイルのほかに，一緒に使う音響モデル定義ファイル *`hmmdefs_file`* が必要です（HTK ASCII 形式 / Julius バイナリ形式のどちらも可）.

Julius で使用する際には，通常のテキスト形式と同じく"`-hlist`" オプションで指定します．テキスト形式かバイナリ形式かの判定は Julius 側で自動的に行われます.

mkbinhmmlist は gzip 圧縮されたファイルをそのまま読み込めます.

Options

`hmmdefs_file` 音響モデル定義ファイル．HTK ASCII 形式，あるいは Julius バイナリ形式.

`HMMList_file` 変換対象の HMMList ファイル.

`output_binhmmlist_file` 出力先となる Julius 用バイナリ形式 HMMList ファイル．すでにある場合は上書きされる.

EXAMPLES

HMMList ファイル `logicalTri` をバイナリ形式に変換して `logicalTri.bin` に保存する：

```
% mkbinhmmlist binhmm logicalTri logicalTri.bin
```

SEE ALSO

julius (1) , mkbinhmm (1)

COPYRIGHT

LICENSE

C.7 adinrec

名前

adinrec – 1 発話の音声入力データをファイルに記録する

Synopsis

`adinrec` [*`options`*...] *`filename`*

Description

adinrec は，音声区間を一定時間内の零交差数とパワー（振幅レベル）のしきい値に基づいて切り出し，ファイルに記録する．デフォルトでは標準デバイスを用いてマイク入力から録音するが，`-input` オプションでデバイスを選択可能である．またプラグイン入力も選択できる．

サンプリング周波数は任意に設定可能である．録音形式は 16bit, 1 channel であり，書き出されるファイル形式は Microsoft WAV 形式である．既に同じ名前のファイルが存在する場合は上書きされる．

ファイル名に "-" を指定すると取り込んだ音声データを標準出力へ出力する．この場合データ形式は RAW 形式になる．

Options

Julius の全てのオプションが指定可能である．指定されたもののうち，音声入力に関係するオプションのみ扱われる．以下に，adinrec 独自のオプションと関係する Julius オプションに分けて解説する．

adinrec specific options

`-freq` ***`Hz`*** 音声のサンプリング周波数 (Hz) を指定する．(default: 16,000)

`-raw` RAW ファイル形式で出力する．

Concerning Julius options

`-input {mic|rawfile|adinnet|stdin|netaudio|esd|alsa|oss}` 音声入力ソースを選択する．音声波形ファイルの場合は `file` あるいは `rawfile` を指定する．起動後にプロンプトが表れるので，それに対してファイル名を入力する．`adinnet` では，adintool などのクライアントプロセスから音声データをネットワーク経由で受け取ることができる．`netaudio` は DatLink のサーバから，`stdin` は標準入力から音声入力を行う．`esd` は，音声デバイスの共有手段として多くの Linux のデスクトップ環境で利用されている EsounD daemon から入力する．

`-lv` ***`thres`*** 振幅レベルのしきい値．値は 0 から 32767 の範囲で指定する．(default: 2000)

`-zc` ***`thres`*** 零交差数のしきい値．値は 1 秒あたりの交差数で指定する．(default: 60)

`-headmargin` ***`msec`*** 音声区間開始部のマージン．単位はミリ秒．(default: 300)

`-tailmargin` ***`msec`*** 音声区間終了部のマージン．単位はミリ秒．(default: 400)

`-zmean` 入力音声ストリームに対して直流成分除去を行う．全ての音声処理のの前段として処理される．

-smpFreq ***Hz*** 音声のサンプリング周波数 (Hz) を指定する．(default: 16,000)

-48 48kHz で入力を行い，16kHz にダウンサンプリングする．これは 16kHz のモデルを使用しているときのみ有効である．ダウンダンプリングの内部機能は sptk から移植された． (Rev. 4.0)

-NA ***devicename*** DatLink サーバのデバイス名 (`-input netaudio`).

-adport ***port_number*** `-input adinnet` 使用時，接続を受け付ける adinnet のボート番号を指定する．(default: 5530)

-nostrip 音声取り込み時，デバイスやファイルによっては，音声波形中に振幅が "0" となるフレームが存在することがある．Julius は通常，音声入力に含まれるそのようなフレームを除去する．この零サンプル除去がうまく動かない場合，このオプションを指定することで自動消去を無効化することができる．

-C ***jconffile*** jconf 設定ファイルを読み込む．ファイルの内容がこの場所に展開される．

-plugindir ***dirlist*** プラグインを読み込むディレクトリを指定する．複数の場合はコロンで区切って並べて指定する．

ENVIRONMENT VARIABLES

ALSADEV (マイク入力で alsa デバイス使用時) 録音デバイス名を指定する．指定がない場合は "`default`".

AUDIODEV (マイク入力で oss デバイス使用時) 録音デバイス名を指定する．指定がない場合は "`/dev/dsp`".

PORTAUDIO_DEV (portaudio V19 使用時) 録音デバイス名を指定する．具体的な指定方法は adinrec の初期化時にログに出力されるので参照のこと．

LATENCY_MSEC Linux (alsa/oss) および Windows で，マイク入力時の遅延時間をミリ秒単位で指定する．短い値を設定することで入力遅延を小さくできるが，CPU の負荷が大きくなり，また環境によってはプロセスや OS の挙動が不安定になることがある．最適な値は OS やデバイスに大きく依存する．デフォルト値は動作環境に依存する．

SEE ALSO

julius (1) , adintool (1)

COPYRIGHT

LICENSE

C.8 adintool

名前

adintool – 音声波形データの記録・分割・送信・受信ツール

Synopsis

`adintool` -in *`inputdev`* -out *`outputdev`* [*`options`*...]

Description

adintool は，音声波形データ中の音声区間の検出および記録を連続的に行うツールです．入力音声に対して零交差数と振幅レベルに基づく音声区間検出を逐次行い，音声区間部分を連続出力します．

adintool は adinrec の高機能版です．音声データの入力元として，マイク入力・ 音声波形ファイル・標準入力・ネットワーク入力 (adinnet サーバーモード) が選択できます．Julius の `-input` オプションも使用可能で，プラグイン入力も選択できます．

出力先として，音声波形ファイル・標準出力・ネットワーク出力 (adinnet クライアントモード) が選択できます．特にネットワーク出力（adinnet クライアントモード）では， julius へネットワーク経由で音声を送信して音声認識させることができます．

入力音声は音声区間ごとに自動分割され，逐次出力されます．音声区間の切り出しには adinrec と同じ，一定時間内の零交差数とパワー（振幅レベル）のしきい値を用います．音声区間開始と同時に音声出力が開始されます．出力としてファイル出力を選んだ場合は，連番ファイル名で検出された区間ごとに保存します．

サンプリング周波数は任意に設定可能です．録音形式は 16bit, 1 channel で，書き出されるファイル形式は Microsoft WAV 形式です．既に同じ名前のファイルが存在する場合は上書きされます．

Options

Julius の全てのオプションが指定可能である．指定されたもののうち，音声入力に関係するオプションのみ扱われる．以下に，adintool のオプション，および有効な Julius オプションを解説する．

adintool specific options

`-freq` *`Hz`* 音声のサンプリング周波数 (Hz) を指定する．(default: 16,000)

`-in` *`inputdev`* 音声を読み込む入力デバイスを指定する．"mic" でマイク入力，"file" でファイル入力，"stdin" で標準入力から音声を読み込む．ファイル入力の場合，ファイル名は起動後に出てくるプロンプトに対して指定する．また，"adinnet" で adintool は adinnet サーバーとなり，adinnet クライアントから音声データを tcp/ip 経由で受け取る．ポート番号は 5530 である（"`-inport`" で変更可能）．

入力デバイスは，そのほか Julius の "-input" オプションでも指定可能である．その場合，プラグインからの入力も可能である．

`-out` *`outputdev`* 音声を出力するデバイスを指定する．"file" でファイル出力，stdout で標準出力へ出力する．ファイルの場合，出力ファイル名はオプション "`-filename`" で与える．出力ファイル形式は 16bit WAV 形式である．また，"adinnet" で adintool は adinnet クライアントとなり，adinnet サーバへ取り込んだ音声データを tcp/ip 経由で送信できる．送信先ホストは "`-server`" で指定する．ポート番号は 5530 である（"`-port`" で変更可能）．

`-inport` *`num`* 入力が adinnet の場合 (-in adinnet)，接続を受けるポート番号を指定する．指定しない場合のデフォルトは 5530 である．

`-server` [host] [,host...] 出力が adinnet の場合 (-out adinnet)，送信先のサーバ名を指定する．複数ある場合は，カンマで区切って指定する．

`-port` [num] [,num...] 出力が adinnet の場合 (-out adinnet)，送信先の各サーバのポート番号を指定する．指定しない場合のデフォルトは 5530 である．`-server` で複数のサーバを指定している場合，全てについて明示的にポート番号を指定する必要がある．

`-filename` *`file`* ファイル出力 (`-out file`) 時，出力ファイル名を与える．デフォルトでは，検出された音声区間検出ごとに，"`file.0000.wav`","`file.0001.wav`" ... のように区間ごとに連番で記録される．番号の初期値は 0 である（`-startid` で変更可能）．なお，オプション `-oneshot` 指定時は最初の区間だけが "`file`" の名前で保存される．

`-startid` *`number`* ファイル出力時，記録を開始する連番番号の初期値を指定する．(デフォルト：0)

`-oneshot` 最初の音声区間が終了したら終了する．

`-nosegment` 入力音声の音声区間検出（無音による区切りと無音区間のスキップ）を行わない．

`-raw` RAW ファイル形式で出力する．

`-autopause` 出力が adinnet の場合（-out adinnet），音声区間が終了するたびに入力停止・動作停止状態に移行する．出力先の adinnet サーバから動作再開信号がくると音声入力を再開する．

-loosesync 出力が adinnet（-out adinnet）で複数の出力先サーバへ出力している場合，動作停止状態から動作再開信号によって動作を再開する際，adintool はすべてのサーバから動作再開信号を受けるまで動作を再開しない．このオプションを指定すると，少なくとも１つのサーバから再開信号がくれば動作を再開するようになる．

-rewind ***msec*** 入力がマイクのとき，停止状態から動作を再開するとき，停止中から持続して音声入力中だった場合，指定されたミリ秒分だけさかのぼって録音を開始する．

Concerning Julius options

-input {mic|rawfile|adinnet|stdin|netaudio|esd|alsa|oss} 音声入力ソースを選択する．"-in" の代わりにこちらを使うこともできる（最後に指定したほうが優先される）．esd やプラグイン入力が指定可能である．

-lv ***thres*** 振幅レベルのしきい値．値は 0 から 32767 の範囲で指定する．(default: 2000)

-zc ***thres*** 零交差数のしきい値．値は１秒あたりの交差数で指定する．(default: 60)

-headmargin ***msec*** 音声区間開始部のマージン．単位はミリ秒．(default: 300)

-tailmargin ***msec*** 音声区間終了部のマージン．単位はミリ秒．(default: 400)

-zmean 入力音声ストリームに対して直流成分除去を行う．全ての音声処理のの前段として処理される．

-smpFreq ***Hz*** 音声のサンプリング周波数 (Hz) を指定する．(default: 16,000)

-48 48kHz で入力を行い，16kHz にダウンサンプリングする．これは 16kHz のモデルを使用しているときのみ有効である．ダウンダンプリングの内部機能は sptk から移植された． (Rev. 4.0)

-NA ***devicename*** DatLink サーバのデバイス名 (`-input netaudio`).

-adport ***port_number*** `-input adinnet` 使用時，接続を受け付ける adinnet のボート番号を指定する．(default: 5530)

-nostrip 音声取り込み時，デバイスやファイルによっては，音声波形中に振幅が "0" となるフレームが存在することがある．Julius は通常，音声入力に含まれるそのようなフレームを除去する．この零サンプル除去がうまく動かない場合，このオプションを指定することで自動消去を無効化することができる．

-C ***jconffile*** jconf 設定ファイルを読み込む．ファイルの内容がこの場所に展開される．

-plugindir ***dirlist*** プラグインを読み込むディレクトリを指定する．複数の場合はコロンで区切って並べて指定する．

ENVIRONMENT VARIABLES

ALSADEV (マイク入力で alsa デバイス使用時) 録音デバイス名を指定する．指定がない場合は "`default`".

AUDIODEV (マイク入力で oss デバイス使用時) 録音デバイス名を指定する．指定がない場合は "`/dev/dsp`".

PORTAUDIO_DEV (portaudio V19 使用時) 録音デバイス名を指定する．具体的な指定方法は adinrec の初期化時にログに出力されるので参照のこと．

LATENCY_MSEC Linux (alsa/oss) および Windows で，マイク入力時の遅延時間をミリ秒単位で指定する．短い値を設定することで入力遅延を小さくできるが，CPU の負荷が大きくなり，また環境によってはプロセスや OS の挙動が不安定になることがある．最適な値は OS やデバイスに大きく依存する．デフォルト値は動作環境に依存する．

EXAMPLES

マイクからの音声入力を，発話ごとに "data.0000.wav" から順に記録する：

```
% adintool -in mic -out file -filename data
```

巨大な収録音声ファイル "foobar.raw" を，音声区間ごとに "foobar.1500.wav" "foobar.1501.wav" ... に分割する：

```
% adintool -in file -out file -filename foobar -startid 1500
% enter filename->foobar.raw
```

ネットワーク経由で音声ファイルを転送する (区間検出なし)：

```
(sender) % adintool -in file -out adinnet -server receiver_hostname -nosegment
(receiver) % adintool -in adinnet -out file -nosegment
```

マイクからの入力音声を Julius へ送信して認識：

```
(sender) % adintool -in mic -out adinnet -server receiver_hostname
(receiver) % julius -C ... -input adinnet
```

SEE ALSO

julius (1) , adinrec (1)

COPYRIGHT

LICENSE

C.9 mkss

名前

mkss – スペクトルサブトラクション用のノイズスペクトル計算

Synopsis

`mkss` [*options*...] filename

Description

mkss は，スペクトルサブトラクション用のノイズスペクトル計算ツールです．指定時間分の音声のない雑音音声をマイク入力から録音し，その短時間スペクトラムの平均をファイルに出力します．出力されたファイルは，Julius でスペクトルサブトラクションのためのノイズスペクトルファイル（オプション"`-ssload`"）として使用できます.

録音は起動と同時に開始します．サンプリング条件は 16bit signed short (big endian), monoral で固定です．既に同じ名前のファイルが存在する場合は上書きします．また，ファイル名に "-" を指定することで標準出力へ出力できます.

Options

-freq ***Hz*** 音声のサンプリング周波数 (Hz) を指定する．(default: 16,000)

-len ***msec*** 録音する時間長をミリ秒単位で指定する（default: 3000）

-fsize ***sample_num*** 窓サイズをサンプル数で指定 (default: 400).

-fshift ***sample_num*** フレームシフト幅をサンプル数で指定 (default: 160).

SEE ALSO

julius (1)

COPYRIGHT

LICENSE

C.10 mkgshmm

名前

mkgshmm – モノフォン HMM を GMS 用に変換する

Synopsis

`mkgshmm` monophone_hmmdefs > outputfile

Description

mkgshmm は HTK 形式の monophone HMM を Julius の Gaussian Mixture Selection (GMS) 用に変換する perl スクリプトです.

GMS は Julius-3.2 からサポートされている音響尤度計算の高速化手法です．フレームごとに monophone の状態尤度に基づいて triphone や PTM の状態を予備選択することで，音響尤度計算が高速化されます.

EXAMPLES

まずターゲットとする triphone や PTM に対して，同じコーパスで学習した monophone モデルを用意します.

次にその monophone モデルを mkgshmm を用いて GMS 用に変換します.

```
% mkgshmm monophone > gshmmfile
```

これを Julius で "`-gshmm`" で指定します.

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

GMS 用モデルは triphone や PTM と同一のコーパスから作成する必要がある点に注意してください．gshmm がミスマッチだと選択誤りが生じ，性能が劣化します.

SEE ALSO

julius (1)

COPYRIGHT

LICENSE

C.11 generate-ngram

名前

generate-ngram – N-gram に従って文をランダム生成する

Synopsis

`generate-ngram` [*options*...] *binary_ngram*

Description

generate-ngram は，与えられた N-gram 確率に従って文をランダム生成するツールです．*binary_ngram* には，バイナリ形式の N-gram ファイルを指定します.

Options

-n ***num*** 生成する文数を指定する（デフォルト：10）

-N 使用する N-gram の長さを制限する（デフォルト：与えられたモデルで定義されている最大値，3-gram なら 3）.

-bos 文開始記号を指定する（デフォルト：<s>）

-eos 文終了記号を指定する（デフォルト：</s>）

-ignore 出力してほしくない単語を指定する（デフォルト：<UNK>）

-v 冗長な出力を行う.

-debug デバッグ用出力を行う.

SEE ALSO

julius (1) , mkbingram (1)

COPYRIGHT

LICENSE

C.12 mkdfa.pl

名前

mkdfa.pl – Julius 形式の文法をオートマトンに変換するコンパイラ

Synopsis

`mkdfa.pl` [*options*...] prefix

Description

mkdfa.pl は Julius の文法コンパイラです．記述された文法ファイル (.grammar) と語彙ファイル (.voca) から，Julius 用の有限状態オートマトンファイル (.dfa) および認識辞書 (.dict) を生成します．カテゴリ名と生成後の各ファイルで用いられるカテゴリ ID 番号との対応が .term ファイルとして出力されます．

各ファイル形式の詳細については，別途ドキュメントをご覧下さい．

`prefix` は，.grammar ファイルおよび .voca ファイルのプレフィックスを引数として与えます．`prefix.grammar` と `prefix.voca` から `prefix.dfa`，`prefix.dict` および `prefix.term` が生成されます．

バージョン 3.5.3 以降の Julius に付属の **mkdfa.pl** は，**dfa_minimize** を内部で自動的に呼び出すので，出力される .dfa は常に最小化されています．

Options

-n 辞書を出力しない．.voca 無しで .grammar のみを .dfa に変換することができる．

ENVIRONMENT VARIABLES

TMP または TEMP 変換中に一時ファイルを置くディレクトリを指定する．指定が無い場合，`/tmp`, `/var/tmp`, `/WINDOWS/Temp`, `/WINNT/Temp` の順で最初に見つかった場所が使用される．

EXAMPLES

文法ファイル `foo.grammar`, `foo.voca` に対して以下を実行することで `foo.dfa` と `foo.voca` および `foo.term` が出力される．

```
% mkdfa.pl foo
```

SEE ALSO

julius (1) , generate (1) , nextword (1) , accept_check (1) , dfa_minimize (1)

DIAGNOSTICS

mkdfa.pl は内部で **mkfa** および **dfa_minimize** を呼び出します．実行時，これらの実行ファイルが，この mkdfa.pl と同じディレクトリに置いてある必要があります．これらは Julius に同梱されています．

COPYRIGHT

LICENSE

C.13 generate

名前

generate – 文法から文をランダム生成する

Synopsis

`generate` [-v] [-t] [-n *num*] [-s *spname*] prefix

Description

generate は文法に従って文をランダムに生成します.
実行には .dfa, .dict, .term の各ファイルが必要です．あらかじめ **mkdfa.pl** で生成しておいて下さい.

Options

-t 単語ではなくカテゴリ名で出力する.

-n ***num*** 生成する文の数を指定する (default: 10)

-s ***spname*** 生成においてスキップすべきショートポーズ単語の名前を指定する．(default: "sp")

-v デバッグ出力.

EXAMPLES

vfr (フィッティングタスク用文法) での実行例：

```
% generate vfr
Reading in dictionary...done
Reading in DFA grammar...done
Mapping dict item <-> DFA terminal (category)...done
Reading in term file (optional)...done
42 categories, 99 words
DFA has 135 nodes and 198 arcs
 -----
silB やめます silE
silB 終了します silE
silB シャツ を スーツ と 統一して 下さい silE
silB スーツ を カッター と 同じ 色 に 統一して 下さい silE
silB 交換して 下さい silE
silB これ を 覚えておいて 下さい silE
silB 覚えておいて 下さい silE
silB 戻って 下さい silE
silB スーツ を シャツ と 統一して 下さい silE
silB 上着 を 橙 に して 下さい silE
```

SEE ALSO

mkdfa.pl (1) , generate-ngram (1)

COPYRIGHT

LICENSE

C.14 nextword

名前

nextword – DFA 文法で（逆向きに）次単語を予測するツール

Synopsis

`nextword` [-t] [-r] [-s *spname*] [-v] prefix

Description

nextword は，**mkdfa.pl** によって変換された DFA 文法上で，与えられた部分文に対して接続しうる次単語の集合を出力します．

実行には .dfa, .dict, .term の各ファイルが必要です．あらかじめ **mkdfa.pl** で生成しておいて下さい．

！注意！ **mkdfa.pl** で出力される文法は，元の文法と異なり，文の後ろから前に向かう逆向きの文法となっています．これは，Julius の第 2 パスで後ろ向きの探索を行うためです．このため，nextword で与える部分文も逆向きとなります．

Options

-t 単語ではなくカテゴリ名で入力・出力する．

-r 単語を逆順に入力する．

-s *spname* スキップすべきショートポーズ単語の名前を指定する．(default: "sp")

-v デバッグ出力．

EXAMPLES

vfr (フィッティングタスク用文法) での実行例：

```
% nextword vfr
Reading in dictionary...done
Reading in DFA grammar...done
Mapping dict item <-> DFA terminal (category)...done
Reading in term file (optional)...done
42 categories, 99 words
DFA has 135 nodes and 198 arcs
-----
wseq > に して 下さい silE
[wseq: に して 下さい silE]
[cate: (NI|NI_AT) SURU_V KUDASAI_V NS_E]
PREDICTED CATEGORIES/WORDS:
            KEIDOU_A (派手 地味 )
            BANGOU_N (番 )
              HUKU_N (服 服装 服装 )
           PATTERN_N (チェック 縦縞 横縞 ...)
              GARA_N (柄 )
             KANZI_N (感じ )
               IRO_N (色 )
             COLOR_N (赤 橙 黄 ...)
wseq >
```

SEE ALSO

mkdfa.pl (1) , generate (1) , accept_check (1)

COPYRIGHT

LICENSE

C.15 accept_check

名前

accept_check – 文法における単語列の受理/非受理チェック

Synopsis

`accept_check` [-t] [-s *spname*] [-v] prefix

Description

accept_check は，文法で文の受理・非受理を判定するツールです．文は標準入力から与えます．受理すべき文を一行ずつテキストファイルにまとめて書いておき，それを accept_check の標準入力に与えることで，その文法 (`prefix.dfa` および `prefix.dict`) において目的の文が受理されるかどうかをバッチ的にチェックできます.

実行には .dfa, .dict, .term の各ファイルが必要です．あらかじめ **mkdfa.pl** で生成しておいて下さい.

対象とする文は，文法の語彙単位 (.voca の第 1 フィールド) で空白で区切って与えます．最初と最後には多くの場合 `silB`, `silE` が必要であることに気をつけて下さい．また，ショートポーズ単語は文に含めないでください.

同一表記の単語が複数ある場合，accept_check はその可能な解釈の全ての組み合わせについて調べ，どれか 1 つのパターンでも受理可能であれば受理，すべてのパターンで受理不可能であれば受理不可能とします.

Options

`-t` 単語ではなくカテゴリ名で入力・出力する.

`-s` ***spname*** スキップすべきショートポーズ単語の名前を指定する．(default: "`sp`")

`-v` デバッグ出力.

EXAMPLES

vfr (フィッティングタスク用文法) での実行例：

```
% accept_check vfr
Reading in dictionary...done
Reading in DFA grammar...done
Mapping dict item <-> DFA terminal (category)...done
Reading in term file (optional)...done
42 categories, 99 words
DFA has 135 nodes and 198 arcs
-----
please input word sequence>silB 白 に して 下さい silE
wseq: silB 白 に して 下さい silE
cate: NS_B COLOR_N (NI|NI_AT) SURU_V KUDASAI_V NS_E
accepted
please input word sequence>
```

SEE ALSO

mkdfa.pl (1) , generate (1) , nextword (1)

COPYRIGHT

LICENSE

C.16 dfa_minimize

名前

dfa_minimize – 有限オートマトン文法を最小化する

Synopsis

`dfa_minimize` [-o *`outfile`*] dfafile

Description

dfa_minimize は，.dfa ファイルを等価な最小化の .dfa ファイルに変換し，標準出力に出力します．オプション `-o` で出力先を指定することもできます.

バージョン 3.5.3 以降の Julius に付属の **mkdfa.pl** は，このツールを内部で自動的に呼び出すので，出力される .dfa は常に最小化されており，これを単体で実行する必要はありません．バージョン 3.5.2 以前の **mkdfa.pl** で出力された .dfa は最小化されていないので，このツールで最小化するとサイズを最適化することができます.

Options

`-o` *`outfile`* 出力ファイル名を指定する.

EXAMPLES

`foo.dfa` を最小化して `bar.dfa` に保存する.

```
% dfa_minimize -o bar.dfa foo.dfa
```

別の方法：

```
% dfa_minimize < foo.dfa > bar.dfa
```

SEE ALSO

mkdfa.pl (1)

COPYRIGHT

LICENSE

Julius の使用許諾に準じます.

C.17 dfa_determinize

名前

dfa_determinize – 有限オートマトン文法を決定化する

Synopsis

`dfa_determinize` [-o *`outfile`*] dfafile

Description

dfa_determinize は，.dfa ファイルを等価な決定性 .dfa ファイルに変換し，標準出力に出力します．オプション `-o` で出力先を指定することもできます.

mkdfa.pl が生成する DFA は常に決定化されており，通常，**mkdfa.pl** で作成された .dfa ファイルに対してこのツールを使う必要はありません.

Options

`-o` *`outfile`* 出力ファイル名を指定する.

EXAMPLES

`foo.dfa` を決定化して `bar.dfa` に保存する.

```
% dfa_determinize -o bar.dfa foo.dfa
```

別の方法：

```
% dfa_determinize < foo.dfa > bar.dfa
```

SEE ALSO

mkdfa.pl (1) , dfa_minimize (1)

COPYRIGHT

LICENSE

Julius の使用許諾に準じます.

C.18 gram2sapixml.pl

名前

gram2sapixml.pl – 認識用文法を SAPI XML 文法に変換するスクリプト

Synopsis

`gram2sapixml.pl` [*`prefix`*...]

Description

gram2sapixml.pl は，Julius の認識用文法ファイル (.grammar, .voca) から Microsoft SAPI XML 形式へ変換するスクリプトです．*prefix* には，変換する .grammar, .voca ファイルのファイル名から拡張子を除外したものを指定します．複数指定した場合，それらは逐次変換されます．

入力文字コードは EUC-JP を想定しています．出力ファイルは UTF-8 エンコーディングです．コード変換のため内部で **iconv** を使用しています．

左再帰性については手作業による修正が必要です．元ファイルの .grammar の構造をそのまま保持するため，.grammar における正順での左再帰記述がそのまま .xml に反映されます．したがって，変換後 .xml に含まれる左再帰性の解決は手作業で行わなければいけません．

SEE ALSO

mkdfa.pl (1)

DIAGNOSTICS

変換は，元ファイルの文法の非終端記号と終端記号 (単語カテゴリ名) をルールに変換するという単純なものです．実際に SAPI アプリケーションで使う場合には，プロパティを指定するなど，手作業での修正が必要です．

内部でコード変換に **iconv** を使用しています．実行パス上に iconv が無い場合，エラーとなります．

COPYRIGHT

LICENSE

Appendix D

利用許諾

Julius はオープンソースソフトウェアであり，無償で利用可能である．使用許諾条件は独自のものであり，概して以下のような特徴を持つ．

- ソースを含まないバイナリのみの配布が可能
- 商用利用を妨げない
- 対外発表時に Julius を利用していることを明記すること

以下に，Julius に付属の仕様許諾条件 LICENSE.txt の内容を転載する．

```
*** English translation is available in the latter of this file ***

「大語彙連続音声認識エンジン Julius」
利用許諾書

  Copyright (c)   1991-2010 京都大学 河原研究室
  Copyright (c)   1997-2000 情報処理振興事業協会 (IPA)
  Copyright (c)   2000-2005 奈良先端科学技術大学院大学 鹿野研究室
  Copyright (c)   2005-2010 名古屋工業大学 Julius 開発チーム

----------------------------------------------------------------------------

「大語彙連続音声認識エンジン Julius」（Julian を含む）は、京都大学 河原
研究室、奈良先端科学技術大学院大学 鹿野研究室、及び名古屋工業大学
Julius 開発チームで開発されています。1997 年度から 3 年間、情報処理振興事
業協会 (IPA) が実施した「独創的情報技術育成事業」の援助を受けました。

京都大学 河原研究室、IPA、奈良先端科学技術大学院大学 鹿野研究室、及び
名古屋工業大学 Julius 開発チーム（以下あわせて「著作権者」と言う）は、
著作者であり著作権を留保していますが、本利用条件の全てを受諾し遵守する
限り、ソースコードを含む本プログラム及びドキュメンテーション（以下あわ
せて「本ソフトウェア」と言う）を無償であなたに提供します。あなたが本ソ
フトウェアを利用したときは、本利用条件の全てを受諾したものと看做されま
す。

【利用条件】

1． あなたは、本利用条件の全てを受諾し遵守する限り、本ソフトウェアの全
部又は一部について使用、複製、翻案、変更、組込み、結合することおよびそ
れらの複製物、翻案物、変更物等を配布、送信することができます。ただし、
あなたを含め本ソフトウェアの利用者は、本ソフトウェアの全部又はその一部
を変更してその複製物を配布、送信などして第三者に提供するときは第 2 項の
表示記載に加え本ソフトウェアを変更した旨、変更者及びその変更日を明確に
表示するものとします。
```

2．あなたは、使用、複製、翻案、変更、組込み、結合その他本ソフトウェアの利用態様の如何にかかわらず、その複製物、翻案物、変更物等の全部又は一部を第三者に提供するときは、本ソフトウェアに下記の著作権表示及び公開の趣旨を含む本利用条件の全て（この文書ファイル）をいささかも変更することなくそのまま表示し添付しなければなりません。

記

3．本ソフトウェアを利用して得られた知見に関して発表を行なう際には、「大語彙連続音声認識エンジン Julius」を利用したことを明記して下さい。

4．本ソフトウェアは、研究開発の試作物としてあるがままの状態で無償公開提供するものであり、本ソフトウェアに関し、明示、黙示を問わず、いかなる国における利用であるかを問わず、また法令により生じるものであるか否かを問わず、一切の保証を行いません。ここで言う保証には、本ソフトウェアの品質、性能、商品性、特定目的適合性、欠陥のないことおよび他の第三者の有する著作権、特許権、商標権等の無体財産権や営業秘密その他の権利利益を侵害しないことの保証を含みますが、それに限定されるものではありません。あなたを含め本ソフトウェアの利用者は、本ソフトウェアが無保証であることを承諾し、本ソフトウェアが無保証であることのリスクを利用者自身で負うものとします。裁判所の判決その他何らかの理由によりあなたに課せられた義務と本利用条件が相容れないときは、本ソフトウェアを利用してはなりません。本ソフトウェアの利用又は利用できないことに関してあなた及び第三者に生じる通常損害、特別損害、直接的、間接的、付随的、派生的な損害（逸失利益を含む）一切につき、それが契約、不法行為責任、瑕疵担保責任、製造物責任等いかなる国のいかなる法律原因によるかを問わず、賠償しません。

5．本ソフトウェアの利用に関しては、日本国の法律を準拠法とし、京都地方裁判所を第一審の専属管轄裁判所とします。

6．本ソフトウェアのメンテナンスやサポート、上記条件以外の利用等に関しては、名古屋工業大学 Julius 開発チーム、または京都大学 河原研究室に照会下さい。

*** This is English translation of the Japanese original for reference ***

Large Vocabulary Continuous Speech Recognition Engine Julius

"Large Vocabulary Continuous Speech Recognition Engine Julius",
including Julian, is being developed at Kawahara Lab., Kyoto
University, Shikano Lab., Nara Institute of Science and Technology,
and Julius project team, Nagoya Institute of Technology (collectively
referred to herein as the "Licensers"). Julius was funded by the
Advanced Information Technology Program Project of
Information-technology Promotion Agency (IPA), Japan for three years

since 1997.

The Licensers reserve the copyright thereto. However, as long as you accept and remain in strict compliance with the terms and conditions of the license set forth herein, you are hereby granted a royalty-free license to use "Large Vocabulary Continuous Speech Recognition Engine Julius" including the source code thereof and the documentation thereto (collectively referred to herein as the "Software"). Use by you of the Software shall constitute acceptance by you of all terms and conditions of the license set forth herein.

TERMS AND CONDITIONS OF LICENSE

1. So long as you accept and strictly comply with the terms and conditions of the license set forth herein, the Licensers will not enforce the copyright or moral rights in respect of the Software, in connection with the use, copying, duplication, adaptation, modification, preparation of a derivative work, aggregation with another program, or insertion into another program of the Software or the distribution or transmission of the Software. However, in the event you or any other user of the Software revises all or any portion of the Software, and such revision is distributed, then, in addition to the notice required to be affixed pursuant to paragraph 2 below, a notice shall be affixed indicating that the Software has been revised, and indicating the date of such revision and the name of the person or entity that made the revision.

2. In the event you provide to any third party all or any portion of the Software, whether for copying, duplication, adaptation, modification, preparation of a derivative work, aggregation with another program, insertion into another program, or other use, you shall affix the following copyright notice and all terms and conditions of this license (both the Japanese original and English translation) as set forth herein, without any revision or change whatsoever.

Form of copyright notice:

Copyright (c) 1997-2000 Information-technology Promotion Agency, Japan
Copyright (c) 1991-2010 Kawahara Lab., Kyoto University
Copyright (c) 2000-2005 Shikano Lab., Nara Institute of Science and Technology
Copyright (c) 2005-2010 Julius project team, Nagoya Institute of Technology

3. When you publish or present any results by using the Software, you must explicitly mention your use of "Large Vocabulary Continuous Speech Recognition Engine Julius".

4. The Licensers are licensing the Software, which is the trial product of research and project, on an "as is" and royalty-free basis, and makes no warranty or guaranty whatsoever with respect to the Software, whether express or implied, irrespective of the nation where used, and whether or not arising out of statute or otherwise, including but not limited to any warranty or guaranty with respect to quality, performance, merchantability, fitness for a particular purpose, absence of defects, or absence of infringement of copyright, patent rights, trademark rights or other intellectual property rights, trade secrets or proprietary rights of any third party. You and every other user of the Software hereby acknowledge that the Software is licensed without any warranty or guaranty, and assume all risks

arising out of the absence of any warranty or guaranty. In the event the terms and conditions of this license are inconsistent with the obligations imposed upon you by judgment of a court or for any other reason, you may not use the Software.

The Licensers shall not have any liability to you or to any third party for damages or liabilities of any nature whatsoever arising out of your use of or inability to use the Software, whether of an ordinary, special, direct, indirect, consequential or incidental nature (including without limitation lost profits) or otherwise, and whether arising out of contract, negligence, tortuous conduct, product liability or any other legal theory or reason whatsoever of any nation or jurisdiction.

5. This license of use of the Software shall be governed by the laws of Japan, and the Kyoto District Court shall have exclusive primary jurisdiction with respect to all disputes arising with respect thereto.

6. Inquiries for support or maintenance of the Software, or inquiries concerning this license of use besides the conditions above, may be sent to Julius project team, Nagoya Institute of Technology, or Kawahara Lab., Kyoto University.