社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－ＳＬＰ－59（22）
2005／12／22

[特別講演] 大語彙連続音声認識エンジン Julius の開発の進展

李　晃伸†

† 名古屋工業大学大学院工学研究科
〒 466-8555 愛知県名古屋市昭和区御器所町
E-mail: †ri@nitech.ac.jp

あらまし　Julius はオープンソースの大語彙連続音声認識ソフトウェアであり，数万語の語彙を用いた連続発声の実時間認識を，一般的な PC 以下のリソースで実現できる．オープンなモデルインタフェース，プログラムの可搬性，および十分な認識処理性能といった特徴を持ち，国内外の様々な研究期間および開発機関で利用されており，現在も実環境における音声インタフェースの手軽な実現と広い普及を目指して，必要な機能の実装と動作の安定性を目指して精力的に開発が続けられている．本稿では，大語彙連続音声認識エンジン Julius の基本アルゴリズムの特徴について解説するとともに，近年の Julius の開発の進展についてまとめる．
キーワード　大語彙連続音声認識，Julius，単語グラフ，GMM，音声インタフェース

Recent Progress of Large Vocabulary Continuous Speech Recognition Engine Julius

Akinobu LEE†

† Graduate School of Engineering, Nagoya Institute of Technology
Gokiso, Showa-Ku, Nagoya, 466-8555 Japan
E-mail: †ri@nitech.ac.jp

Abstract　Julius is a high-performance, two-pass large vocabulary continuous speech recognition (LVCSR) decoder software for speech-related researchers and developers. It realizes almost real-time decoding on most current PCs on dictaion of over 60k words vocabulary. Julius has been developed to equip an open model interface, program portability, and sufficient recognition performance to be used both for the current research of speech recognition and for the development of speech-recognition-driven applications on real world. It has been developed continuously, integrating many required features and improving stability. This paper summarizes the recent progress of the LVCSR engine Julius, with brief description about the search algorithm.
Key words　LVCSR, Julius, word graph, Gaussian mixture model, speech interface

1. はじめに

Julius はオープンソースの大語彙連続音声認識ソフトウェアである．数万語の語彙を用いた連続発声の実時間認識を，一般的な PC 以下のリソースで実現できる．オープンなモデルインタフェース，プログラムの可搬性および十分な認識処理性能といった特徴から，大規模なデータベースに基づく話し言葉認識の研究から，実環境音声対話システム [1]，ヒューマノイドロボット [2]，組み込み用マイコンでの音声認識 [3] など多くの研究および開発機関で利用されている．

開発は 1996 年より京都大学で始まった．最新版は 2005 年 11 月 11 日に公開された rev.3.5 である．1997 年から 2000 年まで IPA の援助を受け，また 2000 年から 2003 年まで情報処理学会連続音声認識コンソーシアム (CSRC)，2003 年から同学会音声対話技術コンソーシアム (ISTC) において頒布が行われている．最新版では実環境での頑健な動作の新機能の追加，性能改善，多くのバグ修正およびソースの大幅な整理が行われた．本体は C 言語で記述されており，ソースコードの行数は，コメントと空行を除いておよそ 33,000 行からなる．

動作プラットフォームは Linux, SunOS, Mac OS X などの Unix 系 OS および Windows である．開発は Linux で行われている．これ以外にも，SH-4 マイコン上への移植 [3] のほか，Windows SAPI への移植やライブラリ化，帯域ごとの尤度に重みをつけられるマルチバンド版の開発 [4] など，他の研究・開発機関や個人による移植や拡張が行われている．一部はボランティアベースで行われ，Web 上から入手可能である．

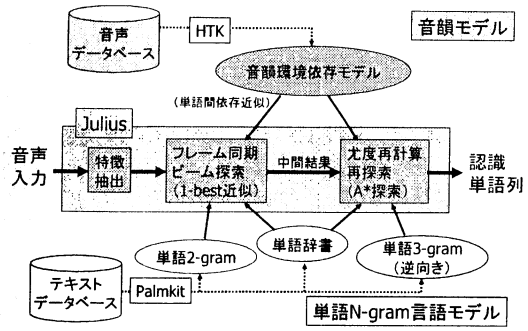

図 1 Julius の 2 パス音声認識アルゴリズムの概観
Fig. 1 Overview of the two-pass recognition algorithm in Julius.

認識のためのモデルとして音韻 HMM と単語辞書，および単語 N-gram を用いる．また記述文法に基づく認識を行うこともできる．この記述文法に基づく機能を持つ Julius は「Julian」と呼ばれ、Julius とソースレベルで統合されている．なお，本稿ではこの Julius と Julian を両方含んだパッケージ全体をまとめて「Julius」と表記する．

Julius で使用可能な言語モデルや音響モデルは，フォーマットが公開されており，モデル構築ツールもフリーで入手可能である．このため，新しい言語でもモデルを構築すれば比較的容易に音声認識を行える．現在多くの言語での動作が報告されており，学会発表ベースでもフランス語やタイ語 [5]，スロベニア語等での動作報告がある．

本稿では，Julius/Julian の認識アルゴリズムの要点についてまとめるとともに，近年の開発の進展についてまとめる．なお，ソースコード，実行バイナリ，解説文書など，多くの情報が Web サイト(注1)から入手できる．また開発中の最新のソースコードは開発サイト(注2)より CVS 経由で入手できる．

2. 音声認識エンジン Julius の概要

Julius は tree-trellis 方式の A*探索に基づく音声認識手法 [6] [7] を，統計言語モデルおよび大語彙向けに拡張した 2 パスアルゴリズムを実装している．ここでは現在の探索アルゴリズムの概観を解説する．

アルゴリズム概要を図 1 に示す．第 1 パスで荒い認識処理を行って可能性のある候補集合を中間結果として出力し，それをもとに第 2 パスで詳細な照合を行う．言語制約として，Julius では単語 2-gram および逆向きの単語 3-gram，Julian では記述文法（有限状態文法）を用いる．単語辞書に各単語のクラス内生起確率を記述することができ，これを利用してクラス N-gram も扱える．音響モデルとしては，トライフォンおよび tied-mixture 形式のモデルも読み込める．なお，トライフォン使用時は，登場しうるすべてのトライフォンから実際に定義されている HMM のモデルへのマッピングを記述した HMMList ファイルが必要である．

2.1 音声入力

音声ファイル，マイクロフォンおよびネットワークから音声を取り込むことができる．扱えるビットレートは 16bit のみである．特徴量は Mel-Frequency Cepstral Coefficients (MFCC)，パワーおよびその一次差分のみサポートしており，それ以外で学習された音響モデルを用いる場合は，あらかじめ HCopy 等で特徴抽出された特徴量ファイル (HTK 形式) として与える．また，サンプリング周波数やウィンドウシフトなどの抽出パラメータを，音響モデルの学習条件と一致するよう明示的に指定する必要がある．

音声区間切り出しは，零交差数および振幅レベルに基づいて行われ，切り出された区間ごとに認識が逐次行われる．ファイル入力では通常 1 ファイルを 1 入力とするが，マイク入力と同様にファイルから音声区間を切り出すこともできる．なお，オプション指定により，音声データを切り出し単位ごとにファイルに記録することができる．

Julius では入力状態に応じた可変バッファリングを行っている．音声検出のために数百ミリ秒のバッファリングを行っているが，非音声区間ではバッファし，音声区間に入るとバッファリングせずに直接入力を認識する状態に移行することで，認識の遅延をできるだけ抑えている．

定常雑音対策としてスペクトルサブトラクションが実装されている．ノイズスペクトルは，各切り出し区間ごとに最初の数百ミリ秒を用いて推定するか，あるいはあらかじめ別ツールで測定したノイズスペクトルを用いることができる．

2.2 第 1 パス

第 1 パスは，木構造化辞書を用いてフレーム同期ビーム探索を行う．言語制約および音響制約の一部を単純化する近似を導入することで，荒いが高速な認識処理を行い，結果を単語トレリスとして出力する．単語トレリスは各フレームにおける終端がビーム内に出現した単語候補リストの集合であり，それぞれ対応する始端フレームおよび累積尤度を保持する．

木構造化辞書は，辞書中の全単語について音素列のプレフィックスを共有して音素 HMM を連結した HMM ネットワークである．ただし同音語については単語末端の 1 状態のみを独立させる．第 1 パスでは，この木構造化辞書上で単語の末尾と先頭のパス遷移を行うことで，仮説空間全体を単一の木構造化辞書のループで効率よく表す．

言語尤度計算においては，単語内の共有ノードで 1-gram 確率の最大値を近似値として割り当てる 1-gram factoring が用いられる．正確な 2-gram factoring に比べ精度が落ちるが，1-gram の最大値は仮説履歴に依存せず，あらかじめ計算可能なため高速な認識が行える．単語が一意に定まった時点で本来の 2-gram 確率を与えるので，結果的に，木構造化辞書上では，各分岐ノードにおいて，それが単語が定まる末端の分岐であれば単語 ID，そうでなければ factoring 値を持つこととなる．この様子を図 2 に示す．

また，木構造化による言語スコアの近似誤差を緩和するため，高頻度単語の線形化を行っている．出現確率の高い単語について，他の単語と共有しないことで，正確な言語スコアを単語先頭で適用する．単語間の言語確率計算では，これらの線形化単語（単語の先頭が factoring ノードでない）についてのみ言語スコア計算を行う．この線形化辞書と単語間言語確率計算の様子を図 3 に示す．

トライフォンモデルを用いる場合，単語末端において，後続可能なすべての単語先頭の音素について異なるトライフォンを

(注1)：http://julius.sourceforge.jp/
(注2)：http://sourceforge.jp/projects/julius/

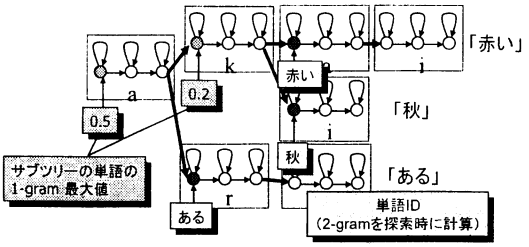

図 2　木構造化辞書における言語スコア表現
Fig. 2　Language scores on the lexicon tree.

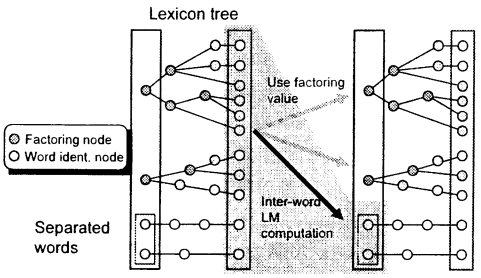

図 3　辞書の線形化と認識時の単語間言語確率計算
Fig. 3　Language scores on the lexicon tree.

個別に計算するのはコストが高いため，単語間の音素環境依存性について近似計算を行う．まず，起動時に音響モデルで定義されているトライフォンのリストから，“k-u+*” のように同じ左あるいは右のコンテキストを持つトライフォン HMM の集合を生成し，それを “k-u” のような仮想バイフォンとして内部で定義する．これを “pseudo phone set” と呼ぶ．木構造化辞書の構築時は，単語の先頭および末尾についてこの pseudo phone set が割り当てられる．第 1 パスの実行時は，単語の先頭および末尾でパスを分けずに最尤のもののみを計算する．単語先頭については，直前単語に従ってトライフォンの状態を切り替えながら尤度計算を行う．単語末尾では，その pseudo phone set に含まれる全てのトライフォンの尤度を計算し，上位候補の平均を近似値として用いる．

2.3　第 2 パス

第 2 パスでは，第 1 パスで得られた単語トレリスを指針として，単語単位のスタックデコーディングによって尤度計算と探索を再び行う．探索の様子を図 4 に示す．探索は音声入力とは逆に入力末端から始端に向かって行われるが，その際に，未探索部分の推定スコア（ヒューリスティック）として単語トレリス上にある各末端単語の始端からの累積スコアを参照することで，仮説の入力全体に対する推定スコアを得ながら best-first に探索を進める．

よく用いられる単語グラフに基づくマルチパス探索では，前段パスで可能性の高い単語終端や仮説単語の接続関係を決定してから後段パスでリスコアリングを行うため，第 1 パスで精度の高い計算を行う必要があり前段パスの計算量が高い傾向にある．これに対して Julius では，登場した仮説をすべて単語トレリスとして後段へ引き渡し，第 2 パスで再計算とトレリスの再接続計算を行う．単語仮説を第 1 パスで固定しないことで，第 2 パスで柔軟な再計算が行え，このため第 1 パスでは高速であるが誤差の大きい計算をも許容できる．

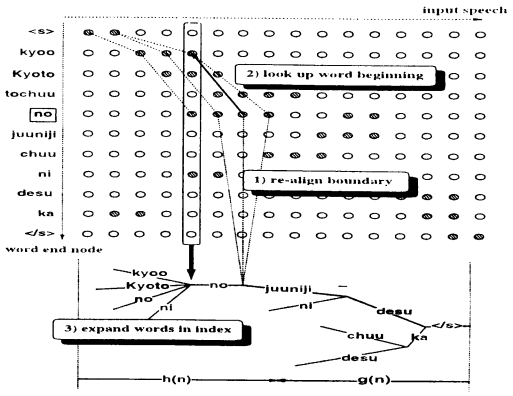

図 4　単語トレリスを用いたスタックデコーディング
Fig. 4　Stack decoding guided by the word trellis.

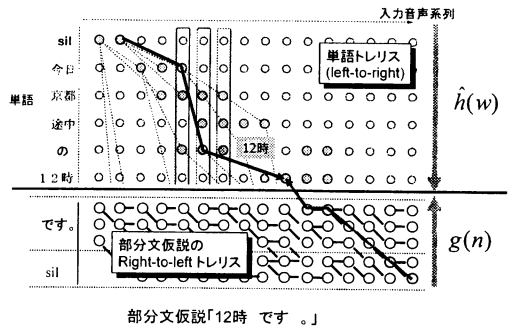

図 5　単語トレリスと部分文仮説の接続によるスコア計算
Fig. 5　Scoring of a partial sentence hypothesis.

部分文仮説の尤度計算におけるトレリス接続の様子を図 5 に示す．展開済みの仮説（図中の仮説の「です 。」）についてこれまでに計算された Viterbi パスの単語先頭部分に，単語トレリス上に存在する次単語（図中の「12 時」）の末端を接続し，その中で最尤の接続点を求める．その後，その部分文仮説が取り出され新たな仮説が生成されるときに，Viterbi パスを 1 単語分更新して新たな仮説を接続する．

探索の安定化と高速化のため，第 2 パスにおいても探索幅制限（ビーム処理）が導入されている．Word envelope beam では，展開仮説数の上限を設け，ある深さにおいてその数以上の仮説展開が起こったらそれより浅い仮説を探索しない．これにより，ある区間で探索が前に進まなくなる現象を回避できる．また，score envelope beam は仮説の Viterbi パス更新計算において，尤度の低いパスを枝仮りして計算量を削減するものであり，フレームごとにそれまでに展開した仮説のフレームごとのスコアの最大値をとり（これをエンベロープと呼ぶ），そこから一定スコア幅のパスのみ計算する．これらのパラメータは実行時オプションで変更可能である．

2.4　On-the-fly decoding

Julius では第 1 パスの認識処理を入力と平行処理することができる．音声区間の開始を検出すると同時に認識処理を開始

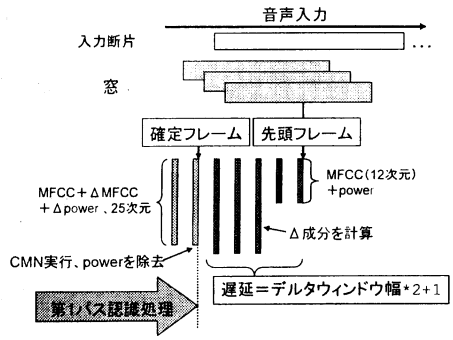

図 6　第 1 パスの on-the-fly 認識処理
Fig. 6　On-the-fly decoding on the first pass.

し，入力音声の特徴量抽出と平行して第 1 パスの認識処理を 1 フレームずつ進める．実装では，スレッドを用いずに，入力音声の読み込みから第 1 パス実行までを巨大な単一のループとして実現しており，入力デバイスから音声断片（デフォルトでは 1000 サンプル）が得られるたびに，特徴量抽出および第 1 パスの認識が漸次的に進められる．実際の並行処理の様子を図 6 に示す．窓掛けおよび特徴量の一次差分の計算のために前後のフレームが必要であるため，認識処理は数フレーム遅延する．

CMN は通常発話単位で行われるが，平行処理時は認識処理時に発話全体のケプストラム平均が得られないため，Julius では直前の 5 秒分の発話のケプストラム平均を用いている．

2.5　逐次音声認識

Julius は，文中の短いポーズで入力を細かく区切りながら，漸次的に認識を行っていくことができる．長い文章の認識に有効であり，特に，文の区切れが明確でなく発話が長時間続く講演音声などの認識に有用である．音声認識では通常，息継ぎや読点など文中に現れる小休止に対応するモデルとして sil, sp などが定義されている．Julius では，第 1 パスで，尤度の一位がこれらの無音モデルを読みとする単語であるようなフレームが一定時間以上続いたとき，そこでポーズ区間が検出されたとして第 1 パスを中断し，その区間に対して第 2 パスを実行する．その後認識を再開するが，その際に前の区間の認識結果を単語履歴として継承する．これによって，逐次に認識結果を確定しながら認識を進める．なお Julian については言語制約の制限からこの機能は使用できない．

2.6　結果出力

認識結果として，単語列，N-gram 単語列，音素列，仮説尤度，単語信頼度等を出力することができる．仮説尤度は音響尤度と言語尤度に分けて出力できる．また，N-best の文候補を出力できる．単語グラフ形式での出力も可能である．

さらに，認識結果に対して forced alignment を行える．Julius では，認識処理中の尤度値は誤差を含むため，認識終了後に改めて単語列に対する Viterbi 計算をやりなおし，正確なアラインメントを出力する．単語単位，音素単位，および HMM の状態単位で，開始フレーム，終了フレームおよび平均音響尤度を出力できる．

3.　単語信頼度および単語グラフ出力

音声認識システムにおいて多様な認識結果の後処理を行うには，一位の文候補のみならず，単語グラフ形式による多数の候補の出力や，認識処理過程に基づく単語信頼度の付与などが有用である．

近年広く用いられている，単語グラフの出力および単語事後確率に基づく単語信頼度の付与 [8] では，正確な事後確率の計算のために入力全体の尤度を用いる必要があり，また，精度良く求めるためには大きなグラフが必要である．このため，典型的な単語グラフ生成は，(1)（1 パスの）認識処理による巨大な単語グラフの生成，(2) グラフ上での単語信頼度の算出，(3) 後処理（リスコアリング，スコアに基づく pruning, confusion network 等による最小化など）といった段階を経ることが多い．

Julius では，第 2 パス中に (1) 探索中のスコアを用いたオンライン信頼度計算，および (2) 動的単語候補確定に基づくオンライングラフ生成 を行うことで，効率良くグラフ生成および信頼度付与を行っている．それぞれ以下に詳細を述べる．

3.1　単語信頼度

Julius では，第 2 パスの探索中に直接単語信頼度を算出する手法を用いている [9]．一般的な単語事後確率は以下の式で求められる．

$$p(w|X) = \sum_{W \in W_{all}} \frac{p(X|W)p(W)}{p(X)} \qquad (1)$$

なお W は文仮説，W_{all} は w を通る全ての文仮説を表す．ここで，$p(X)$ は全ての登場しうる文仮説の出現確率の和として計算できる．ここで，第 2 パスにおいて，式 1 の分子をその時点での（未探索部分に第 1 パスの推定スコアを含む）部分文仮説の尤度，分母をその時点で同時に展開可能な全ての仮説の尤度の合計で近似することで，探索中のスコアから単語展開時に近似的な単語事後確率を求めることができる．多くの単語仮説を残す必要が無いため，高速な計算が可能である．また，探索中に現れる全ての仮説を考慮するため，近似手法ながら巨大なグラフを生成するのと同等の性能が得られている [9]．ただし性能についてはさらなる検証が必要であると考える．

3.2　単語グラフ出力

Julius の rev.3.5 では，[10]．単語仮説の逐次確定および履歴のマージにより，単語グラフを第 2 パス中に動的に生成することができる．この逐次確定の仕組みを図 7 に示す．通常は単語単位で Viterbi 計算を更新するため，仮説の最尤 Viterbi パスは探索終了まで確定しないが，「単語境界はその前後の単語のみに依存する」という単語対近似を導入することで，1 単語前の仮説を順次確定する．

単語確定時に，それまでに確定した単語集合と比較し，同一区間に存在する同一単語があれば互いにマージする．この仕組みを図 8 に示す．これにより，同一の単語履歴がマージされ，動的にグラフを構築する．同一単語であっても前後のコンテキストによって異なる単語境界を持つ場合，マージされずにそれぞれ別々の候補となる．さらに，展開中に部分文仮説の末端が既に確定したグラフの枝と完全に重複する場合，それ以上の仮説展開を中断する．これにより，類似した仮説の展開を抑えて，通常の N-best 探索よりも多様な単語仮説を得る．

従来手法では，まず巨大なグラフを生成してから後処理でリスコアリングや最小化を行う．これに対して本手法は，第 1 パスで仮説を固定せずに単語トレリスの形で多くの単語候補を残し，第 2 パスで仮説の再評価とグラフの生成，最適化を同時に

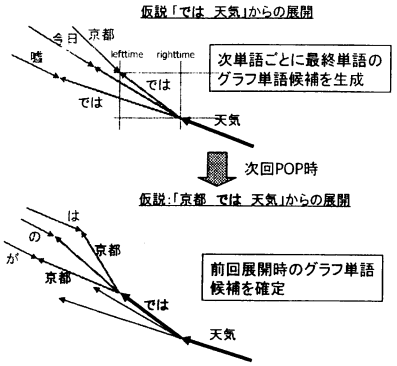

図 7 スタックデコーディングにおける仮説の逐次確定

Fig. 7 Successive determination of word candidates on stack decoding.

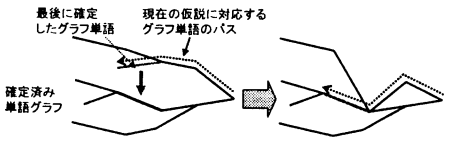

図 8 仮説のマージによる単語グラフの生成

Fig. 8 Generating word graph by merging the contexts on the determinted word tree.

行うものと位置づけられる.

単語グラフ生成時の探索アルゴリズムは，通常時と異なり，最尤の文候補が得られる保証が無い．スコアの高い仮説でもすでに同じ仮説を展開したことがあれば中断されるので，文全体のスコアを最大化するベスト文仮説が得られるとは限らない．また，探索は従来と同じく指定数の文仮説が見つかるまで行われるが，中断により似た仮説が展開されなくなるため，指定数に対して探索は通常のアルゴリズムよりも長くなる．

単語グラフ出力はコンパイル時オプションで有効となる．認識結果として，グラフ中の各単語に対して以下が出力される：

- ID (0 から始まる通し番号)
- 左に接続する単語 ID のリスト
- 右に接続する単語 ID のリスト
- マッチしたフレーム区間
- 単語 ID および表記情報
- 文スコア（g_head + 左単語の第 1 パス尤度）
- 直前の文スコア（g_prev + この単語の第 1 パス尤度）
- 左端状態における第 2 パス累積尤度 (g_head)
- 右端状態における第 2 パス累積尤度 (g_prev)
- 3-gram 尤度
- フレーム平均音響尤度
- 単語信頼度

4. マルチパス版

Julius では，外部からモデル内への遷移，およびモデルから外部への遷移がそれぞれ一つであるという仮定のもと最適化を行っており，そのような遷移を扱えない．

このような音響モデルの制限のないバージョンの Julius が，マルチパス版として提供されている．マルチパス版は，HTK

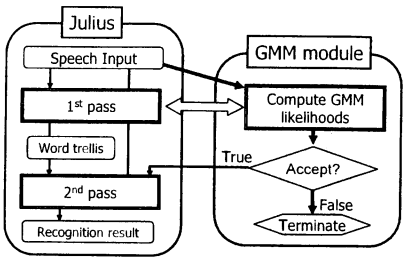

図 9 Julius における GMM による環境音識別と入力棄却

Fig. 9 GMM-based sound discrimination and input rejection on Julius.

で許されるすべての HMM の遷移を扱える(注3)．実装上の大きな違いは，木構造化辞書において全単語の先頭および末端に出力の無い NULL ノードを追加している点である．木構造化辞書のノード数が通常版に比べて大きくなるため，通常版よりも大きいビーム幅を指定する必要がある．また，認識速度は通常版に比べて 1 割から 2 割程度遅い．

マルチパス版にのみ提供される機能として，単語間のショートポーズモデルの付与がある．これは，全単語の末尾にスキップ可能なショートポーズの HMM を付与することで，任意の単語間のショートポーズ（無音区間）をモデル化するものである．

Rev.3.5 より，マルチパス版は通常版のソースに統合された．コンパイル時にオプションを指定することにより，マルチパス版をコンパイルできる．

5. 実環境アプリケーションのための機能強化

5.1 GMM に基づく環境音識別

Rev.3.5 より，Gaussian Mixture Model に基づく環境音識別の機能が組み込まれた．Julius で認識処理と同時に GMM の計算を行い，入力の識別と棄却が行える．GMM は HTK フォーマットで，3 状態（出力 1 状態）の 混合ガウス分布 HMM として与える．できるだけ早期に棄却を確定するために，GMM の尤度計算は Julius の第 1 パスの認識処理と平行して行われる [11]．この仕組みの概要を図 9 に示す．入力フレームごとに認識処理と GMM の計算を行い，入力終了時，すなわち第 1 パスの終了時に GMM の累積尤度が得られる．識別結果が棄却の場合は続く第 2 パスの処理を中断する．また，GMM の尤度計算において，上位分布のみを計算する Gaussian pruning を導入し，実質的な計算量を抑えている．

5.2 複数文法を用いた認識

Julian において，複数の文法を用いて同時に認識することができる．起動時に複数の文法を指定でき，かつサーバー・クライアントモードの際は認識動作中に動的に文法を追加・削除することもできる．

認識時には，その認識結果の属する文法の番号が出力される．また，オプションでそれぞれの文法における最尤仮説を文法ごとに出力することもできる．これは，展開単語を各文法に絞った第 2 パスを各文法ごとに複数回実行することにより行われる．

(注3)：ただし単語全体をスキップするような遷移・辞書の組み合わせのみ許されない．

5.3 サーバー・クライアントモード

Julius はサーバーモードを備えており，他のアプリケーションと TCP/IP 経由で認識結果や命令をやりとりできる．Julius はアプリに対してエンジンの状態，音声入力開始と終了のタイミング，認識結果などの情報を出力する．また Julius に対して音声入力の中断，再開を指示できる．Julian では認識用文法の追加・削除・一時無効化・有効化などが行える．これにより，たとえば文法ベースの音声対話システムにおいて複数のタスク文法を組み合わせた認識を行ったり文法を対話の状態にあわせて切り替えるなど，細かい制御が可能である．

音声認識処理は基本的に音声入力信号によって駆動される処理系であるが，一方でアプリの割り込み命令にしたがって中断や再開を行う必要がある．Julius ではスレッドを行わず，音声入力待ち状態および認識中の各状態においてアプリからの命令キューを定期的にチェックして割り込みを制御している．この機構は Galatea や実環境音声情報案内システム「たけまる君」に用いられている．

6. Julius Rev.3.5

最新版の rev.3.5 は 2005 年 11 月 11 日にリリースされた．実環境音声インタフェースのためのいくつかの重要な新機能の追加および性能改善，安定性の向上とバグ修正，コードの全面的なブラッシュアップが行われた．主な変更点を以下に列挙する．なお認識性能は前バージョン (3.4.2) から変化していない．

- 単語グラフ出力
- GMM による環境音識別および不要音棄却
- 複数文法認識の強化
- メモリ量の改善
- ソースコードの統合とコメント一新
- 英語の利用許諾文書
- 多くのバグ修正
- 入力デバイス追加 (EsounD, ALSA-1.x, Mac OS X)
- 出力文字セット変換 (EUC, SJIS, UTF-8)

第 1 パス処理の冗長な処理の削除，および単語 3-gram のインデックスを 24bit 化したことで，メモリ使用量を 20k 単語で約 20MB 削減した．また，すべてのソースコード内のコメントを書き直した．Doxygen により，全ソースから関数や構造体のクロスリファレンスを HTML 形式で参照できるようになった．基本的に英語で記述し，Julius の認識アルゴリズム部分は日本語と英語を併記している．

本バージョンより，Windows 版および multipath 版のソースコードを通常版に統合した．これより，すべての機能を含む Julius/Julian がひとつのソースアーカイブとして公開される．

7. カスタマイズ

アルゴリズムや多くの機能は，コンパイル時オプションや実行時オプションによって切り替えできる．コンパイル時に選択できるな主なオプションを表 1 に示す．アルゴリズム設定で standard を選択すると，2 音素以下の短い単語について木構造化辞書で線形化を行い，かつ第 2 パスの単語展開時に正確な単語間トライフォンを適用してスコアを計算するようになる．より探索が正確になるが，計算時間が多くかかるようになる．

表 1 主なエンジン設定の組み合わせ
Table 1 Common compile-time options

項目	選択肢
エンジンタイプ	Julius / Julian
音響モデルタイプ	通常 / マルチパス
アルゴリズム設定	fast / standard
出力形式	N-best / 単語グラフ
逐次デコーディング	off / on (Julius のみ)
	(左側がデフォルト)

8. ま と め

大語彙連続音声認識エンジン Julius は，与えられたモデル上で認識を実行する汎用のエンジンとして具備すべき要件として，オープン性，高速性，汎用性，および可搬性を考慮して開発が行われてきた．現在の Julius に実装されている機能は多岐にわたるが，多くは，実環境での様々な音声認識システムの応用において，普遍的に求められる機能であると考える．

今後の予定としては，フロントエンドの耐雑音化，SRGF など文法の拡張，サーバモードの API 拡充などが挙げられる．今後もメンテナンスと開発を続行していく予定である．

文 献

[1] R. Nishimura, Y. Nishihara, R. Tsurumi, A. Lee, H. Saruwatari and K. Shikano: "Takemaru-kun: Speech-oriented information system for real world research platform", Proc. International Workshop on Language Understanding and Agents for Real World Interaction, pp. 70–78 (2003).

[2] "HRP2", http://www.is.aist.go.jp/humanoid/hrp2/hrp2.html.

[3] 小窪, 畑岡, 庄司, 久保, 高橋, 河原, 李, 鹿野："T エンジンに搭載した Julius の高速化を目的とした GMS の計算量削減に関する検討", 音講論, 1-P-6, pp. 163–164 (2005).

[4] 西村, 篠崎, 岩野, 古井："周波数帯域ごとの重みつき尤度を用いた雑音に頑健な音声認識", 電子情報通信学会技術研究報告, SP2003-116, pp. 19–24 (2003).

[5] C. Wutiwiwatchai and S. Furui: "Pioneering a Thai language spoken dialogue system", 音講論, 2-4-15, pp. 87–88 (2003).

[6] F. K. Soong and E. F.Huang: "A tree-trellis based fast search for finding the N best sentence hypotheses in continuous speech recognition", Proc. IEEE-ICASSP, pp. 705–708 (1991).

[7] 河原, 松本, 堂下："単語対制約をヒューリスティックスとする A* 探索に基づく会話音声認識", 電子情報通信学会論文誌, **J77-D-II No.1**, pp. 1–8 (1994).

[8] T. Schaaf and T. Kemp: "Confidence measures for spontaneous speech", Proc. ICASSP, Vol. 2, pp. 875–878 (1997).

[9] A. Lee, K. Shikano and T. Kawahara: "Real-time word confidence scoring using local posterior probabilities on tree trellis search", pp. 793–796 (2004).

[10] 李, 河原, 鹿野："信頼度基準デコーディングを用いた高効率な単語グラフ生成法", 2004-SLP-55, pp. 71–76 (2005).

[11] A. Lee, K. Nakamura, R. Nishimura, H. Saruwatari and K. Shikano: "Noise robust real world spoken dialogue system using GMM based rejection of unintended inputs", Proc. International Conference on Spoken Language Processing, pp. 847–850 (2004).