Julius を用いた音声認識インタフェースの作成

李 晃伸（名古屋工業大学）　河原達也（京都大学）

1　はじめに

生活のさまざまな場面で使用されるユビキタスコンピューティングアプリケーションでは，音声による操作や入力を利用したい場面も多い．そこで，連載第 3 回目では音声認識フリーソフトウェア Julius を紹介する．

Julius はディクテーション（音声入力ワープロのような口述筆記）などの大語彙連続音声認識を目的として筆者らが開発したフリーソフトウェアである．現在も様々な機能拡張や性能改善が継続的に行われており，執筆時点での最新版は 4.1.1 である．関連ファイルとともに，Web サイト（図 1）から無償で入手できる．

Julius Web サイト： http://julius.sourceforge.jp/

本稿では，音声認識の専門家以外の方が Julius を用いて音声認識インタフェースを作成することを想定して，最新版の Julius をベースにその使用法を解説する．以下，2 節で概要を述べた後，3 節で基本的な音声認識の動かし方を説明する．4 節では Julius の動作原理，5 節では設定の方法について簡単に述べる．6 節ではアプリケーションとの連携について，サーバークライアント方式，プラグイン，ライブラリ組み込みの 3 通りの方法を解説する．7 節はトラブルシューティングとして，音声入力や認識がうまく動かない場合に一通りチェックすべきことや気をつけたほうがよい Tips を述べる．

2　音声認識エンジン Julius とは

Julius は大語彙連続音声認識を実行するソフトウェアである．これまでの音声認識の研究で培われてきたアルゴリズムや計算効率化手法をおよそ実装しており，高精度かつ効率よく認識処理を行うことができる．

Julius の最も大きな特徴は汎用性・可搬性にある．音響モデルや言語モデルの仕様は標準的なフォーマットを採用しており，置き換えたり修正したりすることが容易である．これにより，大語彙のディクテーションから小語彙の数字認識まで，様々な使用環境や目的に応じたシステムを構築することが簡単にできる．また言語モデルも，大規模な統計モデルである単語 N-gram，文パターンを人手で与える記述文法，辞書のみを用いる単語認識といったように多様な形式をサポートしており，使用環境に応じて適切なモデルを使い分けることができる．さらに，ソースコードが公開されているので，ユーザ独自の修正や拡張も可能である．

図 1: `http://julius.sourceforge.jp/`

アプリケーションとの連携についても，Julius ではネットワークを介した音声入力や結果のリアルタイムな送受信，プラグインによる機能拡張，認識エンジンライブラリの埋め込みなど，様々な形態をサポートしている．また Microsoft SAPI 対応版も公開されている．逆に，話者の事前登録学習（エンロールメント）や自動適応学習などの機能は現在実装されていない．また，単語を簡単に登録するようなインタフェースも用意されていないため，システム開発者は単語辞書・文法ファイルなどを直接操作する必要がある．

Julius はこのように自由度が高いので，ディクテーションにとどまらず，講演や会議などの自動書き起こし，会話ロボット・エージェント，家電の操作，次世代カーナビなど様々な用途に利用されている．その反面，自由度が高いということは，ある程度中身を理解

して自分で組み立てる必要があることを意味する．例えていうなら，市販のソフトが既製のパソコンのようなものであるのに対して，Julius はカスタムメイドのパソコンに対応する．パーツを理解した上で，自分の目的に合致した仕様のものを選んで構成する必要がある．自分の要求にあう仕様のパーツがなければ，自分で作成することもできるのである（現実的には専門家（音声認識の研究を行っている大学の研究室など）に依頼した方が確実だろう）．

2.1 動作環境

Julius は Windows および Linux をはじめとする Unix 系 OS (Solaris, MacOS X, FreeBSD 等) などのプラットフォームで動作する．Windows では コマンドプロンプト上で動作する．音声入力 API は，Windows では DirectSound，Linux ではカーネル標準の ALSA および OSS, esd をサポートする．（標準モデルを用いる場合）16bit, 16kHz で録音できることが必要である．本体は C 言語で書かれており，Windows 上では MinGW または Cygwin 上でコンパイルできる．

必要なメモリ量の目安は，後述するディクテーション実行キット（2 万語のディクテーション）でおよそ 60 MByte，5,000 語の単語認識で 20MByte 程度である．なお，設定で速度重視か精度重視かチューニングを行うこともできる．

2.2 パッケージ構成

ソースコードおよび 2 種類のバイナリパッケージ（Linux 用および Windows 用）が公開されている．それぞれ Julius 本体および関連ツール，リリースノートおよび開発履歴，man 形式のマニュアル，サンプルコード等が含まれている．バイナリパッケージにはコンパイル済みの実行バイナリが格納されており，すぐに実行することができる．

ドキュメントの多くは，パッケージと別に Web で公開されている．全ての機能と仕様を解説した “Julius-book” という文書（HTML / PDF），チュートリアルや関連文書，およびソースコード中のコメントから Doxygen によって自動生成されたソースコード解説書が Web 上から閲覧できる．

Julius を動かすには音響モデルと言語モデルが必要である．音響モデルは音素ごとの周波数パターンのモデルであり，言語モデルは語彙（単語の発音（音素列）辞書）および単語間の接続制約（文法または単語 N-gram；孤立単語認識の場合は不要）を与える．タスクに合わせてこれらを必要に応じて作成し，Julius と組み合わせることで，任意の認識システムが構成できる．モデルについては 4 節で詳しく述べる．日本語の標準的なモデルは，Web ページの「ディクテーション実行キット」から入手できる．

2.3 開発経緯と派生版

当初 Julius は IPA のプロジェクトである「日本語ディクテーション基本ソフトウエア」(1997〜2000）のコアエンジンとして作成された．したがって，ディクテーションで一般的な単語 N-gram モデル（N 単語連鎖の統計量に基づくモデル；詳細は 4.2 節参照）のみを扱う仕様であった．参考文献 [1] の付録 CD-ROM に収められているものもこの版 (Rev.3.1) である．

一方，音声対話システムなど，扱う発話の範囲が明確な場合は専用の文法を人手で記述することが多い．このような記述文法向けの音声認識エンジンとして，Julian が作成された．こちらは IPA プロジェクトの後継である連続音声認識コンソーシアム (http://www.lang.astem.or.jp/CSRC/) で配布されていたが，2007 年末に公開された Julius-4.0 では Julius で直接記述文法を扱えるようになったため，Julian は Julius に吸収された．このような経緯から，一部の文書では文法に関する部分に “Julian” の呼称が残っている．

Julius にはいくつかの派生版が存在する．Windows SAPI 版は Microsoft Speech API (SAPI) 用の認識エンジンとして動作し，SAPI-5.1 にほぼ完全に準拠している．そのため，アプリケーションは SAPI 経由で Julius を制御できる．ただし Unix 版の後追いで実装しており，3.5 以降の機能は現在のところ使えない．マルチバンド版 Julius は音響特徴ベクトルの各次元に重みづけが行える改造版であり，別途公開されている．使用目的に応じて適当なものを選択されるのが望ましいが，インタフェースの研究開発用としてはオリジナル版を推奨する．

```
% ./bin/julius -C fast.jconf
include config: fast.jconf
（中略）
------------- System Info begin -------------
Julius rev.3.5 (fast)
（システム情報の出力，中略）
------------- System Info end -------------

************************************************************
* NOTICE: The first input may not be correctly recoginized *
*         since no CMN parameter is available on startup.  *
************************************************************

pass1_best:  今 から 温泉 に 旅行 行い ます 。
sentence1:  今 から 音声 入力 を 行い ます 。
<<< please speak >>>
```

図 2: Julius の動作例

3　基本的な使用法

3.1　とりあえずディクテーションソフトとして使ってみる

ディクテーションの実行に必要なもの（Julius 実行バイナリ，N-gram 言語モデル，及び標準的な音響モデル）を１つにまとめた「ディクテーション実行キット」が Web サイトで公開されている．ダウンロードし，展開したディレクトリ上で

```
% ./bin/julius -C fast.jconf
```

と実行することで，マイクからの音声入力に対するディクテーションが行える．Windows 版の場合は代わりに `run_fast.bat` を実行してもよい．動作例を図 2 に示す．モデルをすべて読み込んで起動した後，入力音声ごとに認識を実行する．“`pass1_best`” で始まる行が第 1 パスでの中間結果，“`sentence`” で始まる行が第 2 パスを行った後の最終的な認識結果である．単語列のほかに N-gram エントリ列，音素列，仮説スコア，そして単語毎の信頼度を出力することもできる．ここで，”`-C`” で指定しているのは認識時の設定ファイルである．詳細は 5 節を参照されたい．

マイク入力の音声を認識する場合，Julius はマイクデバイスの設定を行わないため，録音デバイスの選択や録音ボリュームの調節を別途行う必要がある．一部のサウンドドライバでは録音品質が著しく低く認識が困難な場合もある．また，マイク入力においては音響特徴量の正規化（CMS:ケプストラム平均減算）を前の発話に基づいて行うため，最初の発話に対しては認識精度が低下する場合がある．

最新版の Julius を使いたい場合は，`bin` 以下にある実行バイナリを最新版のものと入れ替えればよい．うまく認識しない場合は，7 節を参照のこと．

yomi ファイル（ひらがな読み）“`sample.yomi`”：

```
再生        さいせー
プレイ      ぷれー
停止        てーし
ストップ    すとっぷ
次          つぎ
前          まえ
戻る        もどる
次の曲      つぎのきょく
前の曲      まえのきょく
エンコード  えんこーど
```

dict ファイル（音素列，Julius 用）“`sample.dict`”：

```
再生        s a i s e:
プレイ      p u r e:
停止        t e: sh i
ストップ    s u t o q p u
次          ts u g i
前          m a e
戻る        m o d o r u
次の曲      ts u g i n o ky o k u
前の曲      m a e n o ky o k u
エンコード  e N k o: d o
```

図 3: 単語辞書の例（オーディオ機器操作タスク）

このディクテーションは，タスクを限定せず全ての発話を書き下すことを目的とした認識システムである．一方，特定の応用を考える場合，そのタスクで現れうる語彙や文パターンを限定してシステムに記述する単語認識や文法認識のほうが扱いやすい．次節以降では，単語認識および文法認識の使い方を説明する．

3.2　単語認識をやってみる

孤立単語認識は，機器の操作やロボットへの命令などでは，最もてっとり早い方法である．また，対象を１単語に絞るため計算が単純で処理が速いという利点もある．

孤立単語認識を行うには，そのタスクで認識対象とする単語の辞書を作成する．単語辞書の例を図 3 に示す． まず，単語の出力文字列，およびその読み（ひらがな表記）を 1 行に 1 単語ずつ記述した yomi ファイルを作成する．その後，パッケージに含まれるスクリプト “yomi2voca.pl” で読みを音素列とする dict ファイルに変換する．

```
% yomi2voca.pl sample.yomi > sample.dict
```

```
a i u e o a: i: u: e: o: N w y j
my ky dy by gy ny hy ry py
p t k ts ch b d g z m n s sh h f r
q sp silB silE
```

図 4: 音素表

単語認識を実行するには，Julius に dict ファイルと音響モデルを指定して起動する．音響モデルはディクテーションキットのものを使えばよい．以下のように-h, -hlist オプションで音響モデルファイルを，-w で dict ファイルを指定する．

```
% julius -w sample.dict \
 -h $kit/model/phone_m/hmmdefs_ptm_gid.binhmm \
 -hlist $kit/model/phome_m/logicalTri \
 -input mic
```

起動後は，ディクテーションのときと同様に，音声入力に対して最も可能性の高い単語が出力される．なお，語彙数の上限は 65535 である（コンパイル時の設定で変更可能）．

辞書を作成する際には，読みを実際の発声に沿って記述するようにする．例えば,「携帯」は実際には「けいたい」ではなく「けーたい」,「東京」は「とうきょう」ではなく「とーきょー」と発音されるので，単語辞書上でもそれぞれ後者のように記述する．

参考までに，標準モデルで使用している日本語の音素表を図 4 に示す．ここで，“sp”, “silB”, “silE” はそれぞれ単語間，文頭，文末の無音部分に対応する無音モデルの名前である．これらは単語認識では自動的に扱われるので辞書上で明示的に書くことはない．なお，無音モデル名が上記と異なる音響モデルを使う場合，無音モデル名を “-wsil” で設定する必要がある．

3.3 文法を書いてみる

単語よりも長い文章を認識する場合は，文パターンに関する制約を与える必要がある．ディクテーションでは単語 N-gram を用いるが，天気の情報案内や商品の注文，簡単な音声対話システムといったような，比較的小語彙で想定発話が定型的である場合は，発話パターンを人手で記述する文法認識のほうが使いやすい．

Julius で用いる文法の形式は独自のものであり，文法規則を grammar ファイルに，単語辞書を voca ファイルにそれぞれ記述する．図 5 に，果物注文タスクにおける想定文を受理する文法と語彙ファイルの例を示す．なお，図中の NS_B, NS_E は，切り出された音声入力区間の冒頭と末尾に含まれる無音部分にそれぞれ対応する．

grammar ファイル（文法規則） “fruit.grammar”：

```
S       :  NS_B FRUIT_N PLEASE NS_E
FRUIT_N :  FRUIT
FRUIT_N :  FRUIT NUM KO
FRUIT_N :  FRUIT WO NUM KO
PLEASE  :  WO KUDASAI
PLEASE  :  KUDASAI
PLEASE  :  NISHITE KUDASAI
PLEASE  :  DESU
```

voca ファイル（語彙辞書） “fruit.voca”：

```
% 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
...
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
```

図 5: 文法の例（果物注文タスク）

これらを付属のコンパイラ “mkdfa.pl” を用いて Julius で使用できる形式 (.dict 及び .dfa) へ変換する．Julius には他に文法構築を支援するツールが含まれており，例えば “generate” で与えられた文法に従って文をランダムに生成することで，文法が目的通りの文を受理できるかどうかチェックできる．認識実行時は Julius に “-gram” で指定する．文法コンパイルの過程も含めた動作例を図 6 に示す．設定方法や出力はディクテーションの場合とほぼ同じである．

このような文法を記述するのは慣れないと難しいかもしれないが，Julius の Web サイトにいくつかのタスクや日時や金額など典型的なサブタスクに対するサ

```
【文法のコンパイル】
% 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]
9 categories, 8 nodes, 10 arcs
-> minimized: 8 nodes, 10 arcs
---

【文生成によるテスト】
% generate fruit
Stat: init_voca: read 20 words
Reading in term file (optional)...done
9 categories, 20 words
DFA has 8 nodes and 10 arcs
-----
 <s> リンゴ 4 個 です </s>
 <s> 蜜柑 です </s>
 <s> 蜜柑 8 個 です </s>
 <s> リンゴ 1 個 です </s>
 <s> リンゴ です </s>
 <s> ぶどう です </s>
 <s> 蜜柑 を ください </s>
 <s> ぶどう を ください </s>
 <s> リンゴ 9 個 です </s>
 <s> ぶどう 6 個 です </s>

【認識の実行】
% julius -gram fruit
  -h $kit/model/phone_m/hmmdefs_ptm_gid.binhmm
  -hlist $kit/model/phome_m/logicalTri
  -input mic
（中略）
sentence1: <s> リンゴ 3 個 です </s>
wseq1: 7 0 1 2 6 8
phseq1: silB | r i N g o | s a N | k o | d e s u | silE
cmscore1: 1.000 0.990 0.994 1.000 1.000 1.000
score1: -2832.298340
```

図 6: 文法のコンパイル・テスト・認識の実行

ンプル文法が用意されているので，参考にされたい．

4 動作原理

ここまでの解説で，一通りの使用はできると期待される．ただし，Julius の動作原理を理解した上で，jconf ファイル等で Julius のオプションを適切に設定することで，さらに性能をチューンしたり別の環境に適用することが可能である．以下に Julius の構成を簡単に解説する．

図 7 に Julius の構成を示す．認識処理では，まず入力の音声波形から特徴抽出が行われ，特徴量ベクトルの時系列が抽出される．この入力の特徴量ベクトル系列を X とするとき，Julius は与えられたモデルのもとで最も出現確率が高い単語列 $\hat{W}$ を算出する．すなわち，

$$\begin{aligned}\hat{W} &= \arg\max_{W} P(W|X) \\ &= \arg\max_{W} P(X|W)P(W)\end{aligned}$$

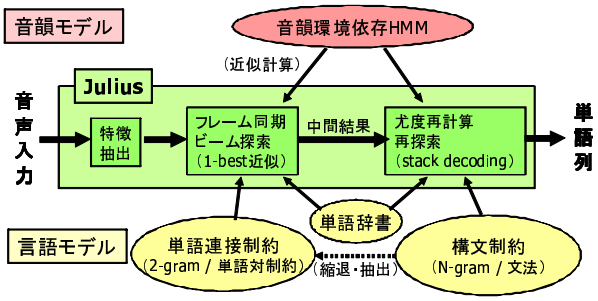

図 7: Julius の構成

この式で，ある単語列 W における X の出力確率 $P(X|W)$ を与えるのが音響モデル，単語列の出現確率 $P(W)$ を与えるのが言語モデルである．

4.1 音響モデル

音響モデル（または音韻モデル）は音素や音節ごとのパターンを保持する．音声認識では HMM (Hidden Markov Model) が主流になっており，Julius も HMM をサポートする．さらに，音素のパターンが先行音素と後続音素に依存するとしてモデル化する音素環境依存モデル（トライフォンモデル）も扱える．トライフォンを用いる場合，単語辞書上の音素表記から得られるトライフォン名と音響モデル上で定義されているモデル名とのマッピングを指定する．前節で “`-h`” で指定したファイルが音響モデル，“`-hlist`” で指定していたのがこのマッピングファイルである．

音響的特徴は，話者（性別・年齢層など）や入力環境（パソコンの接話マイク・電話・自動車内など）によって大きく影響されるので，利用する条件に適合した音響モデルを使用する必要がある．高齢者モデルや子供向けモデル，電話帯域モデルなど，典型的なものについては，連続音声認識コンソーシアム (http://www.lang.astem.or.jp/CSRC/) の成果物として有償で入手可能である．自力で HMM を学習するためのツールキットとして HTK (http://htk.eng.cam.ac.uk/) があるが，音響モデルの構築には言語モデル以上に学習データの収集が大変であり，またかなりの専門的知識も必要とする．

4.2 言語モデル

言語モデルは単語辞書と単語間の接続制約からなる．単語辞書は認識対象の語彙（＝単語の集合）とその発音を規定し，ここで規定されているもののみがマッチングの対象となる．文字認識と異なり，文字を認識してから単語を照合するのでなく，単語辞書を照合しながら文字を認識することに注意されたい．これは，音声が本質的に続け字のように文字の区切りが明確でないためである．

接続制約は，単語 N-gram と文法が使える．単語 N-gram は統計モデルの一種であり，任意の単語の生起確率を，その直前の $(N-1)$ 単語に対する条件付き確率としてコーパス等のテキストデータから学習したものである．Julius は任意の長さの N-gram をサポートし，第 1 パスではその中の 2-gram のみが用いられる．

単語辞書と言語モデルは通常アプリケーションによって規定される．例えば天気案内やホテル検索などのシステムではそれらに特化したものを用意する．汎用的なディクテーション用のモデルは，日本語の大規模なテキストデータベースから構築しているので，それを例えばホテル検索システムにそのまま用いても，固有名詞がカバーされなかったり言語表現の予測能力が十分でないためにあまり高い性能は期待できない．

文法の記述が難しいような複雑度の大きいタスクにおいては，専用の N-gram モデルを用意するのが望ましい．N-gram モデルを学習するためのツールキットとして，CMU-Cambridge 統計言語モデルツールキット (`http://mi.eng.cam.ac.uk/~prc14/toolkit.html`) や palmkit (`http://palmkit.sourceforge.net/`)，SRILM ツールキット (`http://www.speech.sri.com/projects/srilm/`) がある．ただし学習に際しては，アプリケーションで想定されている（あるいは実際に発話された）文のテキストを大量に（数千文以上）収集する必要がある．

4.3 デコーディング

Julius による音声認識の処理は 2 パスで構成されており，第 1 パスでは 2 単語間の接続関係を取り出した単純な言語制約の元，近似による荒い認識処理を高速に行う．第 2 パスでは第 1 パスで得られた仮説集合の情報を参照しながら逆向き（入力末端から先頭に向かって）に厳密な計算を行いながら単語単位の解探索を行い，最終的な候補を決定する．このように，段階的に探索を行い徐々に絞り込んでいくことで，効率の良い認識を行っている．

しばしば誤解されるが，Julius は，与えられた音響モデルや言語モデルに基づいて最尤仮説候補を算出するのみであり，それゆえに（単体では動作しない）認識エンジンと呼ばれる．最尤解を規定するのはあくまでモデルであり，認識精度は基本的にエンジンではなくモデルによって決まる．ただし，大語彙では，設定パラメータによっては過度な近似や計算量抑制により最尤仮説の発見に至らず，精度が下がることもある．

5 設定方法

Julius はモデルの指定のみならず，探索の幅の指定や出力形式の選択など音声認識に関する様々なパラメータをチューニングすることができる．

Julius の設定は，起動時にコマンドラインオプションを指定することで行う．ただし，必要なオプションを全てコマンド引数として書くのは煩雑であるため，通常はテキストファイル内にオプション指定をまとめて書いておき，それを “-C ファイル名” で与える．このファイルは Jconf ファイルと呼ばれる．

オプションで設定できる項目は，全体オプション（オーディオ入力，音声区間検出，デバッグに関する設定），言語モデルオプション（単語 N-gram，文法，単語辞書の指定），音響モデルオプション（HMM 音響モデルの指定，特徴量抽出のパラメータ設定），認識処理・探索オプション（各パスの探索パラメータや言語重み，アルゴリズムの選択），アプリケーションオプション（認識結果の出力形式，文字コード変換，ログ出力，モジュールモードなど，アプリケーションとしての機能に関するもの）など多岐にわたる．Juliusbook の付録 B に全てのオプション（バージョン 4.1.1 で 155 個）の完全な一覧と解説があるので参考にされたい．

6 アプリケーションとの連携

音声認識を用いたインタフェースやアプリケーションを考えるとき，音声認識部とどのように連携・統合するかが問題となる．Julius は当初スタンドアロンの

```
<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>
.
```

図 8: 認識サーバから送信される情報の例

音声認識エンジンとして開発が行われてきたが，近年はアプリケーションとの様々な連携方式をサポートしている．以下でそれぞれについて説明する．

6.1 モジュールモード

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，クライアントアプリケーションと TCP/IP で接続し，認識結果や音声イベントをリアルタイムに送出し，またクライアントからの命令を受信して処理する．

“-module” をつけることで Julius をサーバーモード（モジュールモード）で起動できる．その後，パッケージに含まれるサンプルクライアント “jcontrol” を使って，Julius に接続できる．接続後，音声入力を行うと認識結果や音声イベントが XML 形式でクライアントに随時送信される．例を図 8 に示す．<RECOGOUT>...</RECOGOUT>が認識結果である．認識失敗時は，<RECOGOUT> ... </RECOGOUT> の代わりに<RECOGFAIL/> が出力される．そのほか，エンジンの起動・中断（<STARTPROC/>, <ENDPROC/>），1 回の認識処理の開始・終了（<STARTRECOG/><ENDRECOG/>）など様々な情報がリアルタイムに出力される．

クライアントからはコマンドを送ることができる．jcontrol のプロンプトに pause と入力すると，Julius を一時停止できる．一時停止した Julius は resume で再開できる．他に，文法や辞書を Julius に送ったり，複数の文法を用いた同時認識を制御するなど，多くの機能が提供されている．

詳細な通信仕様は Juliusbook の第 10 章にある．jcontrol のほかに，perl 版のサンプルクライアント jclient.pl もソースパッケージに同梱されている．独自のクライアントアプリケーションを作成する際に参考にされたい．

6.2 プラグイン

最新版の Julius では，プラグインによる拡張をサポートしている．音声入力デバイスを新たに追加したり，認識結果に対する処理を付け加えたりといった機能を追加することができる．

プラグインは拡張子が .jpi のファイルである．その実体は，特定の関数名を外部参照可能に（エクスポート）した共有オブジェクト（Windows では DLL）である．Julius は指定された場所にある .jpi ファイルを動的ライブラリとして読み込み，認識処理中に特定の名前の関数をそれぞれに決まったタイミングで呼び出す．

簡単なプログラム例 test.c を図 9 に示す．このプラグインでは 2 つの関数を定義しており，result_best_str は認識が終わるたびに一位の認識候補の文字列を引数として呼び出される．get_plugin_info はプラグインの種類や説明文字列を返す必須関数であり，起動時に情報取得のため 1 回だけ呼ばれる．コンパイルは，例えば gcc を使う場合，以下のように行える．

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

Julius 実行時に以下のように “-plugindir” でプラグインの置き場所を指定することで，そこにあるプラグインが全て読み込まれる．

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

他にも，音声入力拡張や特徴量入力拡張など，様々な機能拡張が行える．詳細は Juliusbook の 11 章，およびパッケージの plugins 以下にあるサンプルコードを参照のこと．プラグインは実装されたばかりの機能でサンプルが少ないが，要望があればさらなる拡張を行う予定である．

6.3 ライブラリ JuliusLib の利用

バージョン 4.0 より，認識処理部の本体（エンジン）は JuliusLib と呼ばれるライブラリにまとめられた．

```c
#include <julius/juliuslib.h>
void status_recready(Recog *recog, void *dummy) { ... }
void status_recstart(Recog *recog, void *dummy) { ... }
void output_result(Recog *recog, void *dummy)  { ... }
int main(int argc, char *argv[]) {
  Jconf *jconf = j_config_load_file_new(jconf_filename);
  Recog *recog = j_create_instance_from_jconf(jconf);
  callback_add(recog, CALLBACK_EVENT_SPEECH_READY, status_recready, NULL);
  callback_add(recog, CALLBACK_EVENT_SPEECH_START, status_recstart, NULL);
  callback_add(recog, CALLBACK_RESULT, output_result, NULL);
  j_adin_init(recog);
  j_open_stream(recog, NULL);
  j_recognize_stream(recog);
  j_recog_free(recog);
}
```

図 10: julius-simple.c の構成の概要

```c
#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);
  }
}
```

図 9: プラグインのサンプルプログラム

このライブラリをリンクすることで，Julius エンジンを組み込んだ独自の音声認識アプリケーションを作成できる．組み込まれたエンジンとアプリケーションは，コールバックを通じてやりとりする．

サンプルプログラム julius-simple.c がパッケージに同梱されている．構成の概要を図 10 に示す．メインの処理の流れは以下のようになっている．まずエンジンの設定を jconf ファイル（`jconf_filename`）から読み込む．続いて認識エンジンインスタンス `Recog` の生成，コールバックの登録（`callback_add()`），音声入力の初期化を行った後，`j_recognize_stream()` で音声認識のメインループに入る．メインループ中は，JuliusLib 内の認識エンジンから，音声入力開始・終了や結果出力などの各タイミングで対応するコールバックが呼び出される．入力ストリームが最後まで処理が終わると終了する．

コールバック（以下 CB）には以下の種類がある．

認識結果 CB 認識結果に対して実行される．認識の途中経過やアラインメントも得られる．

イベント CB 起動・音声入力検出・認識処理開始／終了などのイベントに対して呼ばれる．

割り込み CB 一定時間おきに繰り返し呼ばれる．アプリ側の定期的なイベントチェックに用いる．

停止時処理 CB エンジン一時停止時の処理（後述）．

音声フェッチ CB 入力音声をモニタできる．

動作中のエンジンの一時停止・再開は以下の要領で行う．あるコールバック内で関数 `j_request_pause()` を実行しておくと，その後音声認識が一時停止し，上記の停止時処理コールバックが呼ばれる．その停止時処理コールバックが終了すると，再び音声認識が動作を再開する．

JuliusLib のコールバックの一覧や API リファレンスは，Web ページの「ドキュメント」から「マニュアル・ソース資料」にある「ソースドキュメントブラウザ」内にある．まだ体系的なドキュメントがないため，利用者が少ないが，今後も改善していく予定である．

7　うまく動かないとき

Julius は基本的に研究開発用ソフトウェアであり，自由度が大きい反面，市販ソフトのような使い勝手がない．うまく動かない場合は，以下の順でチェックしてみるとよい．

7.1　録音状態のチェック

背景雑音やマイクの特性，録音デバイスの種類や設定によって録音品質は大きく変化し，認識性能に直結する．標準の音響モデルは静かな収録環境で性能の比較的高いマイクで録音されたデータをもとに学習されており，雑音レベルやマイクの歪みが大きいと，モデルとのミスマッチにより認識率は悪くなる．まずは，正しく録音できているかをチェックするとよい．

付属のツール adinrec は音声入力を 1 発話分ファイルに録音するツールである．録音および音声区間切り出しは，Julius と同じライブラリを使用している．これで音声を録音し，サウンドレコーダー等でチェックする．ボリュームは，小さすぎると Julius が音声の開始を検出できず，大きすぎると音割れが生じて正しく認識できない．Julius はボリューム調整を行わないので別途調整する必要がある．

マイクの性能は認識精度を大きく左右する．使用する音響モデルにもよるので一般的なことは言えないが，なるべく特性がフラットで，高域まで録音できるものが望ましい．サウンドカードもノイズが混入しにくいものを用いるのが良い．なお，Julius は 48kHz から 16kHz へのダウンサンプラーを内蔵しており，48kHz サンプリングをサポートするデバイスであれば，“-48” を指定することで 16kHz にダウンサンプリングしながら認識できる．特に USB デバイスで有効である．

7.2　音声区間検出のチェック

Julius は振幅の大きさと零交差数をもとに音声入力の開始・終了を判定する．音声区間がうまく切り出せない場合は，検出用のしきい値が適切かどうかチェックする．ボリュームや背景雑音の大きさに合わせて，非発話時と発話時の平均振幅の間にくるように検出用の振幅のしきい値 (“-lv” オプション）を調整する．

7.3　パラメータチューニング

Julius の計算用パラメータは，精度と速度のバランスを取るよう初期設定されている．デフォルトの値で通常問題ないが，探索のビーム幅 (“-b”, “-b2”)，音響尤度計算の計算量 (“-tmix”) が小さすぎると認識がうまくいかない場合がある．

7.4　発話様式

現在の音声認識技術は，誰がどうしゃべっても高い認識率が得られるというわけではない．ある程度機械を意識して，明瞭かつ丁寧に話すことを前提としている．これは，外国人に話しかけるのと同様である．例えば，我々日本人が英語を聞き取る際も，相手が日本人と意識して話してくれると聞き取れるが，外国人どうしの日常的な会話を聞き取るのは非常に困難である．ただし，音声認識ソフトの場合は，単調に話す方がよく，声を張り上げたり,「こーんーにーちーはー」のように強調して発声すると逆効果である．

また，文法的に正しい文を適当に区切って発声する必要がある．

会議や会話のような自然な話し言葉を対象とした音声認識の研究も行われているが，個別のタスクに応じて大規模なデータを収集する必要があり，一般的なモデルはない．

7.5　その他の情報，バグ報告など

Julius の Web ページでは開発者フォーラムを設けており，バグ報告や不具合，応用に関する情報交換が行われている．是非活用いただきたい．なお質問の際は，起動時の標準出力の “System Info begin” から “... end” の部分を一緒に報告いただけると，問題同定に大いに助けになる．

8　おわりに

音声は人間の最も基本的なコミュニケーション手段であり，音声インタフェースは様々な可能性を秘めている．本稿では，音声認識の専門家ではない方が Julius を用いて音声認識インタフェースを構築する際の，道筋や必要な道具立て，注意すべき点などを，最新版の

Julius をベースとしてまとめた．本稿が，音声インタフェース構築への興味，あるいは音声インタラクションの可能性の発見のきっかけとなれば幸いである．

参考文献

[1] 鹿野清宏, 伊藤克亘, 河原達也, 武田一哉, 山本幹雄 編著: 「音声認識システム」, オーム社, (2001) (ISBN 4-274-13228-5).

[2] 李 晃伸:「大語彙連続音声認識エンジン Julius ver.4」情報処理学会研究報告，2007-SLP-69-53, (2007)

[3] 河原達也, 李晃伸.「連続音声認識ソフトウエア Julius」人工知能学会誌, Vol.20, No.1, pp.41–49, (2005).

[4] 河原達也, 荒木雅弘. 音声対話システム. オーム社, (2006).