第7章 言語モデル

目次

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．

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．

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．

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．

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．

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．

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
```

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
```

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．

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．

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．

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
```

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．

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．

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．

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．

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．

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．

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．

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．

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
```

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．

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
```

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>
```

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
```

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．

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．

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．

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．

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．