ネットワーク文法の作成方法

李　晃伸

平成 15 年 2 月 27 日

1　連続音声認識パーザ Julian について

音声認識モジュールは，連続音声認識パーザ Julian に基づいて作成されました．Julian は有限状態文法 (DFA) に基づく連続音声認識を行うソフトウェアです。与えられた文法規則の元で入力音声に対して最尤の単語系列を探しだし出力します．

Julian を用いた音声認識システムは，図 1 のように**タスク文法**と**音響モデル**を Julian に与えることで構成されます．タスク文法は，その認識システムで扱う入力文のパターンを与えます．音響モデルは通常音素単位で与えられ，各単語の音響尤度計算に用られます．

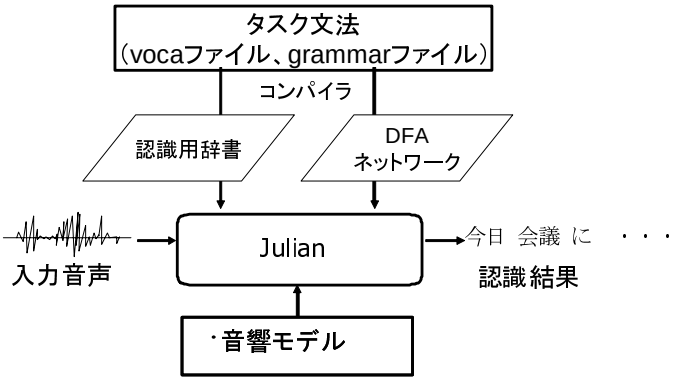
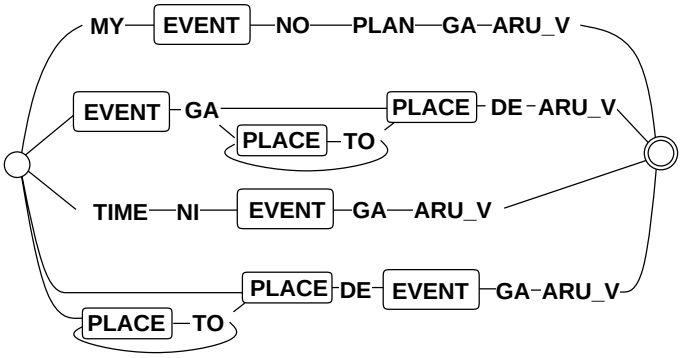


図 1: Julian を用いた音声認識システム

タスク文法　タスク文法は，語彙を登録する単語辞書 **voca ファイル** と，各単語の構文上の接続関係を記述した **grammar ファイル** からなります．システム設計者は，対象とするタスクで発声される内容を受理できるようこれらのファイルを記述します．これらを専用コンパイラ “mkdfa.pl” を使って，それぞれ Julian 用の認識辞書と有限状態オートマトン (DFA) ネットワークに変換して使用します．図 2 に DFA ネットワークの例を示します．Julian は入力に対して，この DFA ネットワーク上で最尤の系列を見つけて出力します．なお Julian の扱える最大語彙数は 65,535 語です．



（各シンボルは単語のカテゴリを表す）

図 2: DFA ネットワーク

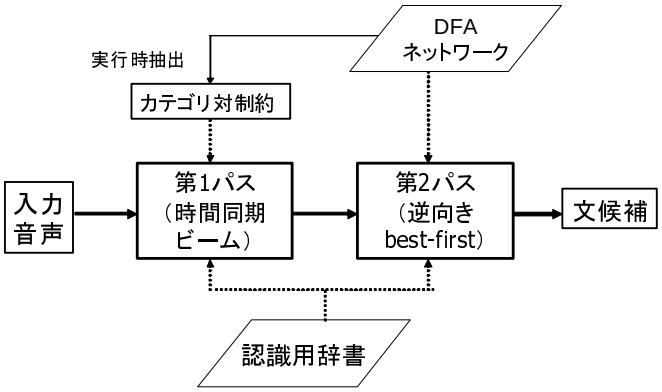


図 3: Julian の解探索アルゴリズムと文法の適用

認識アルゴリズム　Julian の認識アルゴリズムは，2 パス構成の A*探索です．探索アルゴリズム，および各パスでの文法の適用について図 3 に示します．まず第 1 パスでは，処理効率を優先して，DFA ネットワークをさらに縮退させて単語のカテゴリ間の接続関係のみを抽出した「カテゴリ対制約」を用い，高速な時間同期ビーム探索を行います．この第 1 パスは入力と平行して処理されます．第 2 パスではその結果をヒューリスティックとして参照しながら，本来の DFA を用いて動的に展開しながら探索を行い，最終的な認識結果を出力します．

2　文法の記述方法

julian では，タスク文法を，構文制約を単語のカテゴリを終端規則として BNF 風に記述する grammar ファイルと，カテゴリごとの単語の表記と読み（音素列）を登録する voca ファイル に分けて記述します．以下，文法の書式から Julian の起動までを解説します．

2.1 grammar ファイル

grammar ファイルは 1 行に 1 つの書き換え規則を記述します．左辺に書き換えられるシンボル，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します．開始記号は "S" です．英数字とアンダースコアが使用でき，大文字・小文字は区別されます．

左辺に現れなかったシンボル群がこの文法における終端記号，すなわち単語のカテゴリ群となります．各カテゴリに属する単語は voca ファイルで記述することになります．

例として，1 桁の数字を認識する孤立単語認識タスクのための簡単な grammar ファイルを以下に示します．NUM の場所に任意の 1 数字が入ることを想定しています．NS_B と NS_E はそれぞれ文頭および文末の無音に対応する単語カテゴリです．この例の場合，voca で定義すべき単語のカテゴリは NS_B, NUM, NS_E となります．

```
S :  NS_B NUM NS_E
```

なおこの構文制約は形式上 CFG のクラスまで記述可能ですが，Julian はオートマトン，すなわち正規文法のクラスまでしか扱えません．この制限はコンパイル時に自動チェックされます．また再帰性は左再帰性のみ扱えます．

2.2 voca ファイル

voca ファイルでは，grammar ファイルで使用する単語カテゴリごとに単語を登録します．1 行に 1 単語ずつ，表記と発音音素列を登録します．行頭が "%" で始まる行はカテゴリの定義です．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り，grammar で使用するカテゴリが voca で定義されていなかったり，逆に voca で定義されたカテゴリが grammar で現れない場合はエラーとなります．

上記の grammar の例に対する voca ファイルは以下のようになります．

```
% NS_B
<s>         silB
% NS_E
</s>        silE
% 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
8           h a ch i
9           ky u:
```

2.3 Julian 形式への変換

grammar ファイルと voca ファイルは，専用コンパイラ `../julian/mkdfa.pl` を用いて Julian 用の DFA ネットワークファイル (.dfa) と辞書ファイル (.dict) に変換します．以

下はこれまでの例をそれぞれ sample.grammar, sample.voca としたときの実行例です. dfa, dict, term の 3 ファイルが出力されます[1].

```
% ../julian/mkdfa.pl sample
sample.grammar has 1 rules
sample.voca    has 3 categories and 12 words
---
Now parsing grammar file
Now modifying grammar to minimize states[-1]
Now parsing vocabulary file
Now making nondeterministic finite automaton[4/4]
Now making deterministic finite automaton[4/4]
Now making triplet list[4/4]
---
-rw-r--r--    1 foo      users          42 Aug 26 01:22 sample.dfa
-rw-r--r--    1 foo      users         155 Aug 26 01:22 sample.dict
-rw-r--r--    1 foo      users          20 Aug 26 01:22 sample.term
```

2.4 Julian の起動と認識の実行

以下の要領で Julian を起動できます. -h で音響モデル, -dfa で dfa ファイル, -v で dict ファイルを指定します.

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

-input mic でマイク入力の直接認識を行えます. また-input rawfile で音声ファイルの認識も行えます.

-n オプションと -output オプションで, N ベストの数を指定できます. 無指定の場合は 1 位の認識結果が得られます. "-n N -output N" の形で指定します. また-progout で第 1 パスの認識経過を漸次的に出力することができます.

[1] term ファイルはカテゴリ番号とカテゴリ名の対応リストで, Julian では用いません.