日本語テキスト処理のための「かな」コード試案

藤田 毅
九州産業大学工学部
fujita@te.kyusan-u.ac.jp

1 はじめに

日本語はひらがな，カタカナ，漢字など文字種は非常に豊富であるが，音素の数がきわめて少ないいわば文字ベースの言語である．西欧語は，文字種は大文字，小文字のアルファベット文字が基本で非常に少ないが，音素の数は概して日本語よりもはるかに多くいわば音声ベースの言語といえる．

同じ表音文字とされている「かな」(ひらがなとカタカナ) とアルファベットを比較すると,「かな」文字は読み (音節) とほぼ1対1に対応するが，アルファベット文字には直接の対応関係はない．

日本語の読みは50音表で示されるように，ごく単純な2次元性 (子音と母音の組み合わせ) を示す．だから読みとほぼ1対1に対応する「かな」文字にはこの2次元性が内包されていると考えられる．

アルファベットの場合はほとんど識別のための記号に過ぎないので，機械可読のための文字の符号化において，1次元に配置した順序数としても問題はない．しかし「かな」文字を符号化したＪＩＳコードにおいても同様の符号化，すなわち1次元の順序数として構成されているため,「かな」文字が内包している2次元性 (子音と母音) が反映されない．このため，単語をその読みの順にソート (国語辞書式順ソート) するようなごく単純な問題ですら，濁音，半濁音文字を清音文字に変換するなどの特別な処理が必要になる．

日本語の表音文字表記法としての (日本式) ローマ字表記はこの2次元性を備えている．このため，音韻レベルでの処理を必要とする場合に，ローマ字表記に変換して処理する方法もよく用いられる．しかし，単に変換の手間がかかるだけではなく，ローマ字自体も西欧語の表記に準じている側面があるため,「かな」文字の持つ特性を十分には表していない．(すなわち，清音，濁音間の類似性が捕えられない．文字数が1，2，3文字とばらつく，カタカナ語の表記が統一されていない，など.)　本論文では，本来「かな」文字が内包する50音すなわち，子音，母音などの属性情報を含んだ「かな」文字コードを提案する．これを機能的かなコード (Functional Code for Japanese Phonetic Symbol)，あるいは簡単にＦコードと呼ぶことにする．

その基本的な考え方はこれまで自然言語処理での最小情報単位としてきた文字コードの内部を子音属性や母音属性などを示すいくつかのビットフィールド (属性部) に分けることにある．そうすると1文字 (1または2バイト) 毎の照合ばかりでなく文字内部の子音属性あるいは母音属性など部分的な属性情報を照合することも可能になる．このため文字列照合操作においても一致，不一致だけでなく，音韻などの部分一致による不完全文字列検索などが可能になる．

たとえば，文字列 'いちはつ' に対して，'いっぱつ' と 'いそみつ' を比較した場合，音韻属性の一致度まで考慮すれば，'いっぱつ' の方が 'いそみつ' よりも似ていることが直接的に求められる．

更に文字内部の一部の属性値を変更すると音韻的に関連する別の文字に置き換わる．このような書き換え操作はたとえば動詞の活用変形で代表される語と語の連接時の音韻的変形の処理に利用できる．さらに口語表現での音の崩れ (食べてしまう→食べちまう→食べちゃう) やカタカナ語の表記の揺れ (ジェネレーション→ゼネレーション) など，日本語の形態素処理に関わるいくつかの問題に対しても応用可能であることを示す．

2 Ｆコードの構成

2.1 日本語テキストの文字コード

Ｆコードとして取り上げるのは全角の「かな」(ひらがな文字と，カタカナ文字) である．漢字やその他の全角文字は従来のＪＩＳコードをそのまま利用するものとし，それらは各バイトの最上位ビットを1とした2バイトのＥＵＣコードで表すものとする．

半角のASCIIコードについては上位バイトをすべて零とした2バイトコードに変換する．そうすれば全角のひらがな，カタカナに対応するFコードには，最上位のビットが零であり，上位バイトが零ではない2バイトコードを割り当てることができる．

５０音表の行，列をそれぞれ子音属性（C-属性），母音属性（V-属性）に割り当てるとすれば，少なくとも子音は１０個，母音は5個の異なる値を持つ．これらの属性を独立に扱うためには少なくともそれぞれ4ビット，3ビットの領域が必要になる．すると「かな」のもっとも基本的なC-V特性の情報がちょうど上位1バイト分に割り当てられることになり，機械処理上からも都合がよい．

次にマイナーな属性として，弱音（小文字），直音，濁音，半濁音の4つを区別するための音変化属性（I-属性）として2ビット，ひらがな，カタカナの別を示すための文字種属性（T-属性）として1ビットが必要である．この計１０ビットが「かな」文字を識別するための領域となる．

残り5ビットは余りとなるが，この部分は文脈内での文字の音韻的な変化の可能性を指定するための領域（可能変化属性，P-属性）として用いることにする．ここには最大３２通りの異なる変化可能性が指定できることになる．

2.2 Fコードの属性情報

Fコードの各属性のビット割当てを図1に示す.(Fコードのビット位置0の値はつねに零である.)

ここでC-V-I-Tの各属性はそれぞれ以下のような属性値を取る．

C-属性 $= \{0, k, s, t, n, h, m, y, r, w, N, X\}$
子音無し,「か」行,「さ」行,「た」行,「な」行,「は」行,「ま」行,「や」行,「ら」行,「わ」行,「ん」，任意の子音.

V-属性 $= \{0, a, i, u, e, o, X\}$
母音無し,「あ」,「い」,「う」,「え」,「お」,任意の母音.

I-属性 $= \{0, j, b, p\}$
直音，弱音，濁音，半濁音.

T-属性 $= \{kh, kk\}$
ひらがな，カタカナ.

各属性値はそれぞれの集合の中で，昇順の順序関係が成り立つように整数が割り当てられる．

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	*C*				*V*			*I*		*T*	*P*				

図 1: Fコードのビット構成

```
int Fstrcmp(FCODE *fs1, FCODE *fs2)
{
   int diff = 0;

   while(fs1->CV == fs2->CV){
      if( !diff) diff = (fs1->I)-(fs2->I);
      if(fs1->val == FNULL) return diff;
      fs1++; fs2++;
   }
   return (fs1->CV)-(fs2->CV);
}
```

図 2: Fコード文字列比較関数 Fstrcmp

なおカタカナ長音文字‘ー’は，半濁音化された母音文字（「あ」行文字）としてコード化する．

2.3 国語辞書式ソート

Fコードのビット位置0を最上位と見れば，各属性間に次の順序関係がなりたつ．

$C > V > I > T > P$

このような性質を利用した簡単な応用例として，国語辞書式順に並べる問題を取り上げる．

２つのFコード列（へのポインタ）fs1とfs2を国語辞書式で比較するための関数FstrcmpをC言語標準ライブラリの文字列比較関数strcmpの自然な拡張としてコーディングした例を図2に示す．[1]

strcmpとの違いは上位バイト（ＣＶ値）が一致する場合にのみ，下位バイト内（Ｉ値）で最初に現れる差異を保存する点である．

この文字列比較関数を文字列ソートプログラムの中に組み込み,「かな」見出しのテキストファイルをソートした結果を図3に示す．

3 Fコードの照合

以下ではFコード列を明示する場合，次のように‘F’で始まる名前の列で表示する．

$$F_1 F_2 \ldots F_n$$

FコードFの属性・属性値の関係を明示する場合

[1] ここでFCODEはFコードの各属性の組としての2バイト構造体と2バイト符号無し整数のデータ型からなる共用体とする．

（原テキスト）	（ソート後）
% cat ex.text	% jwsort <ex.text
アイシング	ああ
あいしょう	アーケード
アーケード	あい
あいしん	アイ
ああ	あいしよう
アイ	あいしょう
あい	あいじょう
あいしよう	あいしん
あいじょう	アイシング

図 3: 「かな」見出し列のソート

は，次の例のように‘⟨’と‘⟩’の中に記述する．

$F\langle C:0, V:a, I:0, T:kh\rangle \Leftrightarrow$ ‘あ’

ただし，あいまいさのない場合，次のように記述することもある．

あ $\langle C:0, V:a, I:0, T:kh\rangle$

「かな」文字識別のためのFコードの総数は約2^{10}通りであるが，現実の「かな」文字は約１６０個しかない．このため対応文字が存在しないコードに対してはもっとも類似した「かな」文字を割り当てておくこととし，これを**縮退マップ文字**と呼んでおく．たとえば次のFコード（ローマ字の‘wi’）は‘い’に対応させ，縮退マップ文字であることを示すために‘$\bar{い}$’と表すことにする．

$\bar{い}\langle C:w, V:i, I:0\rangle$

また$C:X, V:X$はそれぞれ任意の子音，母音と一致することを示す．

例えば，次の「カ」行子音動詞‘書$\bar{く}$’はその活用形‘書か’，‘書き’，‘書く’，‘書け’，‘書こ’のいずれとも一致する．

書$\bar{く}\langle C:k, V:X, I:0\rangle$

実際には文字照合処理の際，次のように指定された属性の照合のみを行い，他の属性は無視することによって文字の部分照合が実現できる．

$F\langle C:k, I:0\rangle$

Fコードの属性毎の照合機能は文字列同士の音韻的な類似性を求めるのにも利用できる．従来のように文字の一致，不一致だけで判定する場合，たとえば4文字長の文字列‘いちはつ’に対して，‘いっぱつ’，‘いしまつ’，‘いそみつ’はいずれも下線部が一致するので，類似度は$(1+0+0+1)/4=0.5$とすべて同一となってしまう．それに対してFコードでは各属性毎の照合結果の和として文字の一致度を判定することができる．たとえばC-属性を0.5，V-属性を0.4，I-属性を0.1の重みとすれば次のような類似度の相違が求められる．

いっぱつ $= (1+0.5+0.9+1)/4 = 0.85$

いしまつ $= (1+0.5+0.5+1)/4 = 0.75$

いそみつ $= (1+0.1+0.1+1)/4 = 0.55$

4 Fコードの置換

属性αに値βを代入するとき$\alpha := \beta$と表す．これは部分情報による文字の置換操作であり，音韻的に関連する別の文字に置き換わる．たとえばFコードテキスト内ではカナ文字F_1に続くカナ長音F_2‘ー’は次のように直前文字の母音に置き換えられる．

$F_1\langle T:kk\rangle F_2\langle C:0, V:X, I:p\rangle$

$\rightarrow F_1F_2\langle V := V(F_1)\rangle$

ここで‘$V(F_1)$’はF_1のV-属性の値を指示するものとする．

このような置換操作はさまざまな音韻的な変形に対して有用である．ここではその一例として日本語に数多く現れる次のような短縮化現象を考察する．

書く + います ⇒ 書きます （動詞と接辞の連接）

食べてしまう ⇒ 食べちまう （口語表現）

ジェネラル ⇒ ゼネラル （カタカナ語の揺れ）

この種の2字から1字への短縮変形はすべて次の置換操作によって行なわれる．（**連結合成規則**）

$F_1F_2 \rightarrow$

$F\langle C := C(F_1), V := V(F_2), I := I(F_1)\rangle$

このような処理はローマ字表記などを用いても可能ではあるが，Fコードでは母音文字，子音文字の区別なく，単一の規則にまとめられる．

5 音韻変化可能性

音韻変化に伴う日本語の表現のずれはさまざまな形で現れるが，これをFコード列の変形形式の相違として分類してみると，2字から1字への**短縮**，1字内の属性変化による**推移**，および逆に1字から2字への**添加**の3つに大別できる．以下，これを更に詳細に分類した結果を示す．（分類内のイタリック記号はP-属性値を示す．）

(1) 短縮

(1-1) 連結合成（df）

(1-2) 削除（dd）

食べていらっしゃる⇒ 食べてらっしゃる

食べ+いた ⇒ 食べた

ディジタル⇒ デジタル

(2) 推移

(2-1)C-属性推移

1) 拗音化 (cY)

食べちまう ⇒ 食べちゃう

食べては駄目 ⇒ 食べちゃ駄目

2) 撥音化 (cN)

食べたのだ ⇒ 食べたんだ

飛びた ⇒ 飛んだ (撥音便)

3) 促音化 (cQ)

学 (がく) ⇒ 学校 (がっこう)

割りた ⇒ 割った (促音便)

4) 子音削除 ($c0$)

書きた ⇒ 書いた (イ音便)

美しく ⇒ 美しう (ウ音便)

(2-2) V-属性推移

1) a (va)

風 (かぜ) ⇒ 風見 (かざみ)

2) i (vi)

メール⇒ メイル

3) u (vu)

ショー⇒ ショウ

4) e (ve)

威張る (いばる) ⇒ えばる (方言)

5) o (vo)

赤 (あか) う⇒ あこう

6) w (vw), y (vy)

ニカラグア⇒ ニカラグワ (半母音化)

(2-3) I-属性推移

1) 弱音・直音 (ij)

フイルム⇒ フィルム

2) 濁音・直音 (ib)

玉 (たま) ⇒ 白玉 (しろだま)

3) 半濁音・直音 (ip)

発 (はつ) ⇒ 単発 (たんぱつ)

フェイス ⇒ フェース

(3) 添加

(3-1) 区切り添加 (ss)

メモリアクセス ⇒ メモリ・アクセス

(3-2) 長音添加 (sR)

コンピュータ ⇒ コンピューター

(3-3) 撥音添加 (sN)

みな ⇒ みんな

(3-4) 促音添加 (sQ)

ま+しろ ⇒ まっしろ

(3-5) 拗音添加 (sY)

悲しう ⇒ 悲しゅう

(3-6) U 型変化 (sU)

カルテット ⇒ クァルテット

これらの中で, 動詞の活用形に関しては日本語文法においてかなり規則的に扱われている. しかしそれでも従来の方法では活用処理は複雑である [4]. 本論文では詳細は割愛するが, 日本語は活用しない, とする派生文法 [2] に F コードの機能性を組み合わせるとこの処理がかなり簡単化される.

その他の現象は大部分, 個々の語彙や文脈に依存するものでありルール化することは困難である. しかし個別の語彙に依存する変化可能性は, F コードにより語彙見出しにおける対応文字の P-属性として個々に指定しておくことが可能となる.

ただし, 同一文字において複数の変化の可能性がある語彙に関しては, 語彙定義の中に一括して記述せざるをえないが, このような語彙の数はそれほど多くないと思われる. (たとえば変格活用動詞 '来る', 'する', など)

6 おわりに

日本語の「かな」は本来５０音表のような２次元性を内包している. ここで提案した「かな」コードはこれを忠実に１文字２バイトの中に符号化しようとしたものである. そして更にそれらの音韻的な属性情報を独立に取り扱うための機能性を持たせた. これにより文字内部の部分的な属性照合操作や, 属性値の変更による文字の置換操作などが容易に実現できる.

その結果, 従来の自然言語処理における最小情報単位であった文字レベルから, よりミクロなレベルの情報を用いたテキスト処理が可能となることを示した.

参考文献

[1] 長尾真 (監修) : 日本語情報処理, 電子情報通信学会, (1984).

[2] 清瀬義三郎即府: 日本語文法新論–派生文法序説–, 桜楓社, (1989).

[3] 村木新次郎: 日本語動詞の諸相, ひつじ書房, (1991).

[4] 久光徹, 新田義彦: 日本語形態素解析における効率的な動詞活用処理, 94-NL-103-131, 情報処理学会研究会報告 (1994).