Emacsで漢字の読みを残す機能の問題点について

平成 23 年 2 月 10 日

情報電子工学科
大桃 隆宏

目 次

概要

過去の卒業研究で Emacs で漢字の読みを残す機能を実装するという研究があった。これは漢字からひらがなに変換する時の誤変換をさけるために漢字変換時読みとして入力したひらがなを残しておく機能の実装を行なった研究である。しかし、この研究にはいくつか問題点が残されていた。その一つである「難しい漢字の読みを知らない時、または正しい漢字が変換しても出ない時に正しい読みのひらがなを残せない」という内容の問題を解決するのが本研究の目的である。これを解決するために Emacs-Lisp という Emacs 専用の拡張言語で書かれている YC というかな漢字変換プログラムの改良を行なうことを考えた。YC にはひらがなを格納しておく yc-hiragana-list という変数が存在するが、そこで新しく yc-list-back という変数を追加し、yc-hiragana-list に格納したひらがなを yc-list-back にコピーしておき、yc-hiragana-list で正しい読みが残せない時に yc-list-back からひらがなを出力するようにして正しい読みを残すことができるのではないかと考え実験した。いくつかの方法で実験したところ yc-cancel という「変換を取り消す関数」に変更を加えることで正しい読みを残すことができた。しかし、この方法にもいくつか問題点が残っている。

1　はじめに

漢字の含まれる文章を点字や音声で出力するにはひらがなが必要になる。しかし、漢字からひらがなに変換する場合、誤変換が起こる場合がある。その誤変換を避けるため Emacs で漢字変換時に読みとして入力したひらがなを残す機能が過去の卒業研究で実装されたが、いくつか問題点が残されていた。本研究はその問題点を解決するのが目的である。本研究室では、主に Emacs というテキストエディタで文章 を作成している。Emacs は、様々な機能を持っているテキストエディタで Emacs-Lisp という Emacs 専用の拡張言語があり、自由にカスタマイズできる。Emacs-Lisp で作られたかな漢字変換プログラムに YC というものがあり、過去の卒業研究ではこの YC を改良することでひらがなを残すことに成功している。残された問題点を解決する にはこの YC プログラムをさらに改良する必要がある。そのために Emacs-Lisp を勉強し、YC の構造を調べ、問題点の解決策を考察した。

2　使用するソフト

2.1　Emacs

コンピュータで文字情報のみのファイルを作成、編集、保存するためのソフトウェアのことをテキストエディタという。Emacs とは、そのテキストエディタの種類の一つである。レポートの作成や電子メール等の様々な機能を持っているだけでなく、Emacs-Lisp という Emacs 専用の拡張言語があり、自由にカスタマイズできるようになっている。主に、UNIX (OS の一種) を使っている人に人気があるテキストエディタである。

かな漢字変換プログラムの YC では Emacs-Lisp が使われている。Emacs-Lisp とは Emacs が内蔵する拡張言語であり、主に

- Emacs の設定
- 機能の拡張
- Emacs 機能そのものの実装

等に使われている。

利用者は Emacs-Lisp を用いて Emacs を自由に拡張できる。Emacs-Lisp は Emacs の機能拡張に特化した言語であるため、C 言語で書かれた Emacs のソースコードを変更する手段に比べ、手軽に拡張できる。

以下に Emacs-Lisp のコマンドの一部を紹介する。

(+ 1 2 3 4 5)

このように入力する。これは 1,2,3,4,5 の足し算をする式で、ここで ”＋” は関数であり、後に続く引数の和を求めている。

(and [式 1] [式 2].....[式 n])

and は式を順に評価していき、値が nil になったときに、nil を返す。最後まで nil がでないときは、その最後の値を返す。

(or[式 1] [式 2].....[式 n])

or は式を順に評価していき、値が nil でなくなったときに、その値を返す。最後まで nil のときは nil を返す。

(not [式])

not は式の論理を反転する。

(defun 関数名 (引数)
“関数の簡単な説明”
関数の内容)

この defun というコマンドは、新たに関数を定義するコマンドである。Emacs-Lisp ではプログラムを作るときに、この defun という関数を頻繁に使う。

(if [条件] [式 1] [式 2])

Emacs-Lisp では条件を判断する if 文はこのようになる。[条件] の値が nil 以外なら [式 1] を実行し、そうでなければ [式 2] を実行する。[式 2] は省略することも可能である。

(setq x 1)

setq というコマンドは変数自体に値を格納する関数である。上の例では”x” に”1” を格納している。変数に型はないが、値の型に応じて処理が行われる。データ型は関数、数値、リスト、配列などがある。

(insert [文字列])

現在、カーソルがある場所に、[文字列] を挿入する。

(while [条件式] [式])

ループを形成する唯一のコマンドが、while である。[条件式] が nil 以外なら、[式] を実行する。[式] は、複数でも可能である。

2.2 canna

canna(かんな) は日本語入力用のかな漢字変換ソフトである。NEC が開発した日本語入力システムで、Canna という名前は、「かな漢字変換」の「かな (仮名) の古い読み方である「かんな」からによるものである。もともと UNIX 用に開発されたもので、かな漢字変換の方式は、クライアント・サーバ方式となっている。かな漢字変換辞書の種別として、システム辞書、グループ辞書、ユーザ辞書の 3 種類の辞書があり、辞書に対して、辞書の非所有者の読み込み権、所有者の書き込み権といったアクセス権を設定することができるようになっている。Emacs で canna を使うには YC を使用する。

2.3 YC

YC とは Yet another Canna client を省略したもので、 Canna のかな漢字変換をするためのサーバと直接通信をして、かな漢字変換を Emacs 上で実現するプログラムである。

以下に、YC の長所をまとめる。

- Canna 対応していない Emacs でも Canna が使える。
- 字種変換結果が候補に含まれる。例:「研究」と入力し変換すると候補の中に「kennkyuu」の用に字種が違う物も候補に含まれる。
- 確定直後なら再変換できる。候補一覧モードから漢字を選んで確定した時、もう一度確定を押す前なら再変換できる。

YC を実際に使う場合は .emacs に以下のコマンドを追加する。

```
;;
;; for yc mode
;;
(setq yc-use-color t)
(setq yc-use-fence (not (eq window-system 'x)))
(setq yc-server-host "nolm01")

(load "yc")
;(global-set-key "\M-\ " 'yc-mode)
(global-yc-mode 1)
```

(load "yc") を変更を加えた yc のファイルを指定する。

試験的に YC を使用する場合は .emacs をコピーしたテキストファイルを用意する。ここでは仮に yc-emacs.txt とする。そして、kterm 上に emacs-raw -q -l yc-emacs.txt と打ち込んで起動させる。

YC はいくつかのモードにわかれている。

- かな入力モード
  入力されたローマ字をひらがなに変換しながら入力モード
- かな編集モード
  読みを編集したり漢字などに変換する変換の指定をするモード
- 漢字変換モード
  指定された読みを漢字等に変換するモード
- 候補一覧モード
  変換候補の一覧を表示し、そこから選択するモード

以下に各モードに使われている YC の関数の一部とその関数に対するコメントの一覧のを紹介する。

まずは、かな入力モードの中で使われている関数を紹介する。

関数名	関数の説明
yc-input-self-insert	入力されたローマ字をひらがなに変換しながら読みを入力する。

次に、かな編集モードで使われている関数を紹介する。

関数名	関数の説明
yc-edit-jisyu	編集中に字種変換する
yc-edit-katakana	読み編集中にカタカナ変換する
yc-edit-beginning-of-fence	最初の読みに移動する
yc-edit-end-of-fence	最後の読みに移動する
yc-edit-forward-char	読み編集中に次の読みに移動する
yc-edit-backward-char	読み編集中に前の読みに移動する
yc-edit-backward-delete-char	前の読みを削除する
yc-edit-delete-char	カーソルのある読みを削除する
yc-edit-self-insert	読みを入力する (追加入力、挿入等)
yc-edit-henkan	ひらがな漢字変換する (変換の指定)

次に、漢字変換モードで使われている関数を紹介する。

関数名	関数の説明
yc-get-kouho-list	server から変換候補を取得する関数
yc-next-kouho	次の候補を選択する関数
yc-forward-bunsetsu	次の文節を選択する関数
yc-henkan-region	指定された範囲を漢字変換する
yc-kakutei	漢字変換を確定する
yc-cancel	変換を取り消す関数
yc-enlarge	変換中の文節を伸ばす関数
yc-shrink	変換中の文節を縮める関数
yc-forward	次文節に対象文節を移動する
yc-backward	前文節に対象文節を移動する

次に、候補一覧モードで使われている関数を紹介する。

関数名	関数の説明
yc-select	一覧モードを開始する関数
yc-select-forward	一覧モードで次候補に移動する関数
yc-select-backward	一覧モードで前候補に移動する関数
yc-select-nobasi	一覧モードで対象文節長を伸ばす
yc-select-tidime	一覧モードで対象文節長を縮める
yc-select-beginning-of-line	一覧モードで候補群の先頭に移動する関数
yc-select-end-of-line	一覧モードで候補群の末尾に移動する関数
yc-choice	一覧モードで候補を選択する関数
tc-select-cancel	一覧モードを中止する関数

3 過去の研究について

3.1 プログラムの比較

オリジナルの YC プログラムと過去の卒業研究 [1)] で作られた改良された YC プログラムの比較をしていく。

二つのプログラムを比較したところ、漢字変換モードの「確定の処理をする関数」の一部 (yc-kakutei の内部処理をする関数) に変更が加えられていることがわかった。以下は変更を加える前の yc-kakutei-internal である。

```
(defun yc-kakutei-internal ()
  (let ((idx 0)
(yc-mark-max yc-mark-max))
    (while yc-mark-max
      (when (>= (nth idx yc-mark-list) (car yc-mark-max))
(setcar (nthcdr idx yc-mark-list) 0))
      (setq idx (1+ idx))
      (setq yc-mark-max (cdr yc-mark-max))))
  (setq yc-symbol-list (reverse (delq nil yc-symbol-list)))
  (while yc-symbol-list
    (when (cdar yc-symbol-list)
      (yc-put-symbol (string-to-char (caar yc-symbol-list))
     (cdar yc-symbol-list)))
    (setq yc-symbol-list (cdr yc-symbol-list)))
```

```
  (condition-case nil
      (yc-end-convert (yc-get conv)(length yc-henkan-list)\\
        1  yc-mark-list)
    (yc-trap-server-down nil))
  (yc-fence-mode nil)  ; 表示していた文字を消去
  (insert (apply 'concat yc-henkan-list))   ; 漢字を出力
  (set-marker yc-fence-end (point))
  (setq yc-henkan-mode nil
yc-mark-list nil
yc-mark-max nil
yc-kouho-list nil
yc-romaji-list nil
yc-hiragana-list nil)
  (when yc-isearch
    (setq yc-isearch nil)
    (if (featurep 'xemacs)
(isearch-nonincremental-exit-minibuffer)
      (exit-minibuffer) )))
```

そして、以下が改良された YC プログラムの yc-kakutei-internal の一部分である。

```
  (defun yc-kakutei-internal ()
            :
            :
  (yc-fence-mode nil) ; 表示していた文字を消去
  (insert (apply 'concat yc-henkan-list))  ; 漢字を出力
  (insert (apply 'concat yc-hiragana-list)); ←新しく追加
  (set-marker yc-fence-end (point))
            :
            :
```

このようにコマンドが一行追加されているのがわかる。(yc-fence-mode nil) とは入力したひらがなを一度消去するコマンドである。その後、yc-henkan-list に保持されている漢字リストを concat で連結して漢字を出力する。通常はこれで変換は終了だが改良された YC では、最初に入力して yc-hiragana-list に格納してあったひらがなをもう一度出力していることがわかる。

3.2 括弧で括る方式

漢字の隣に括弧で括ったひらがなを出力する。

今日は [きょうは] 天気が悪い [てんきがわるい]

このように、出力したひらがなを括弧で括る場合、YC プログラムを以下のようにする。

```
(defun yc-kakutei-internal ()
          :
          :
(yc-fence-mode nil) ; 表示していた文字を消去
(insert (apply 'concat yc-henkan-list))  ; 漢字を出力
(insert  "[")
(insert (apply 'concat yc-hiragana-list)); ひらがなを出力
(insert  "]")
(set-marker yc-fence-end (point))
          :
          :
```

しかし、括弧で括る場合は問題がある。C 言語等で括弧の中のひらがなだけを取り出そうとすると「が」や「は」といって接続詞等が取り出せない。

3.3 セパレータで括る方式

括弧で括るだけでは、問題がある。そこで、その括弧を違うものにする。文章を作るときに使わないような文字列である必要がある。過去の研究ではセパレータを使用していた。例を以下に示す。

@@@KanjiPart@@@今日は@@@KanaPart@@@きょうは@@@EndPart@@@

このような長いセパレータを使えば他の文章とぶつかる可能性は低くなる。ただし、この場合は見にくくなってしまう。セパレータで括る場合、YC プログラムは以下のようになる。

```
(defun yc-kakutei-internal ()
          :
```

```
          :
  (yc-fence-mode nil) ; 表示していた文字を消去
  (insert "@@@KanjiPart@@@")
  (insert (apply 'concat yc-henkan-list))  ; 漢字を出力
  (insert \begin{quote}
\begin{verbatim}
  (defun yc-kakutei-internal ()
          :
          :
  (yc-fence-mode nil) ; 表示していた文字を消去
  (insert "@@@KanjiPart@@@")
  (insert (apply 'concat yc-henkan-list))  ; 漢字を出力
  (insert "@@@KanaPart@@@")
pp  (insert (apply 'concat yc-hiragana-list)); ひらがなを出力
  (insert "@@@EndPart@@@")
          :
          :
```

3.4 次の行に出力する方式

ひらがなを漢字の隣ではなく次の行に出力する。

今日の天気は雨
きょうのてんきははれ

このように出力する場合プログラムは以下のようになる。

```
  (defun yc-kakutei-internal ()
          :
          :
  (yc-fence-mode nil) ; 表示していた文字を消去
  (insert (apply 'concat yc-henkan-list))  ; 漢字を出力
    (save-excursion
       (forward-line)
       (if (eobp) (insert "\n@@@"))
       (end-of-line)
       (if (bolp) (insert "@@@"))
  (insert (apply 'concat yc-hiragana-list))); ひらがなを出力
```

```
(set-marker yc-fence-end (point))
          :
          :
```

しかし、次の行に出力する場合、ひらがなの読みがどの漢字に対応するか分からないという問題がある。

4 YC プログラムの改良点の考察

4.1 過去に残されていた問題点

過去の研究で以下の問題点が残されていた。

- 違う読みの入力
  難しい漢字の読みを知らない時、または正しい漢字が変換しても出ない時に
  正しい読みのひらがなを残せない
  例えば流鏑馬 (やぶさめ) という漢字がある。この漢字を「やぶさめ」と入力しても正しい漢字は 出てこない。そこで「りゅうかぶうま」と入力して変換すればこの漢字は出てくるがそれだと読 みは「りゅうかぶらうま」と出力されて正しい読みではないという問題点である。
- 「 わ」と読む「は」
  文章を音声で出力する場合に「わ」と「は」の区別が難しい。すべてひらがなであると「わ」と 読む「は」なのか「は」と読む「は」なのか区別が難しいという問題点である。

今回はこの問題点の中で「違う読みの入力」を YC プログラムを改良して解決したいと思う。

4.2 yc-cancel

yc-cancel とは変換を取り消す関数のことである。使い方は漢字の確定前の変換中の状態で C-g を押すことで変換を取り消すことができる。

例：研究 (確定前) → C-g を入力→けんきゅう

以下は yc-cancel のプログラムである。

```
;; 変換を取り消す関数
;; 変換前の状態に戻す
(defun yc-cancel ()
  "漢字変換を中止し、変換前の状態に戻す"
  (interactive)
  (setq yc-mark-list (make-list (length yc-henkan-list) 0))
  (condition-case nil
      (yc-end-convert (yc-get conv)
(length yc-henkan-list) 0 yc-mark-list)
    (yc-trap-server-down nil))
  (if yc-romaji-list
      (progn
(setq yc-henkan-mode nil
      yc-mark-list nil
      yc-mark-max nil
      yc-kouho-list nil
      yc-symbol-list nil)
(setq yc-edit-mode t)
(yc-edit-post-command-function))
    (yc-fence-mode nil)
    (setq buffer-undo-list (primitive-undo 1 buffer-undo-list)
  yc-kouho-list nil
  yc-mark-list nil
  yc-mark-max nil
  yc-henkan-mode nil
  yc-symbol-list nil)
    (set-marker yc-fence-end (point))
    (when yc-isearch
      (setq yc-isearch nil)
      (if (featurep 'xemacs)
  (isearch-nonincremental-exit-minibuffer)
(exit-minibuffer)))))
```

具体的には変換モード時に C-g でフェンスモードに戻るという関数である。

- フェンスモード
  文字を入力した時に確定を押す前まで文字の両側についている”—” をフェンスと

いい、フェンスで 囲まれている場所が現在入力や編集を行なっている文字である。フェンスで囲まれている状態をフ ェンスモード と呼びリターンキーで確定するとフェンスモードが解除される。

「違う読みの入力」という問題点は漢字変換しても正しい漢字が出てこないということなので使用者は必ず yc-cancel を使うのではないかと考えた。この yc-cancel に何らかの改良を加えて取り消したひらがなを残して、それを読みとして使えればこの問題点の解決に近付くのではないかと思う。

4.3 yc-hiragana-list

「違う読みの入力」の問題点を解決するには正しいひらがなを格納しておく必要があると考えた。そのためには入力したひらがなを格納しておく変数が必要である。そこで、YC には最初からひらがなを格納して使っている変数があった。それは、yc-hiragana-list という変数である。現在の入力の状態を、使用者に確認させるために、ひらがなを表示する必要があるので、最初からこのような変数があると思われる。

4.4 どのような改良を加えるか

C-g を入力してフェンスモードからフェンスモードの外 (何も無い状態) まで戻ってしまうと yc-yomi-reset というコマンドが働いて yc-hiragana-list に格納してあったひらがなも消えてしまうようなので新しい変数を用意してそこにひらがなを格納する必要があると考えた。ここでは仮に新しい変数を 「yc-list-back」 とする。

例えば、yc-yomi-reset はフェンスモードからフェンスモードの外 (何も無い状態) まで戻るときに働くが、その前に yc-hiragana-list に格納してあったひらがなを変数 yc-list-back に格納する。その後、新しく入力して漢字変換したひらがなの読みを yc-hiragana-list からではなく yc-list-back から出力するようにすれば「違う読みの入力」に関しての問題は解決するのではないかと思う。

5 YC プログラムの改良

最初に yc-list-back を新しく定義する必要がある。これは yc-hiragana-list が定義されていた「読み入力＆読み編集モード」の冒頭部分に追加する。プログラムは以下のようになる。

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 読み入力&読み編集モード
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; fence モード用変数
; yc-fence-yomi にキー入力したローマ字を格納している
(defvar yc-hiragana-list nil)
(defvar yc-romaji-list nil)
(defvar yc-yomi-string-point nil)
(defvar yc-yomi-list-point nil)
(defvar yc-list-back nil)   ←この行を追加
```

defvar とは新しく変数を定義するコマンドである。この行は yc-list-back の初期化を表している。

5.1 条件分岐なしの改良

入力したひらがなを残すには yc-hiragana-list の中にあるひらがなを yc-list-back にコピーしておく必要があると考えた。そうすれば yc-yomi-reset が働いても格納してあったひらがなを残しておくことが可能である。コマンドで表すと

```
(setq yc-list-back yc-hiragana-list))
```

というコマンドをどこかに追加すれば良いと考えた。

yc-hiragana-list にひらがなを格納しているのは読みを入力した時なので「読みを入力する関数」に上記のコマンドを追加すればいいのではないかと考えた。以下が上記のコマンドを追加した関数である。

```
;; 読みを入力する
(defun yc-yomi-insert (str)
  (setq yc-fence-yomi (concat (substring yc-fence-yomi 0 yc-yomi-string-point)
      str
      (substring yc-fence-yomi yc-yomi-string-point))
yc-yomi-string-point (+ yc-yomi-string-point (length str)))
  (let* ((conv (and (> yc-yomi-list-point 0)
    (string= (nth (1- yc-yomi-list-point) yc-hiragana-list)
```

```
      (nth (1- yc-yomi-list-point) yc-romaji-list))))
  (res (if conv
   (yc-conv-rH-list
    (concat (nth (1- yc-yomi-list-point) yc-romaji-list) str))
 (yc-conv-rH-list str)))
  (dif (if conv -1 0)))
     (setq yc-hiragana-list
   (append (yc-subsequence yc-hiragana-list 0
   (+ yc-yomi-list-point dif))
   (car res) (nthcdr yc-yomi-list-point yc-hiragana-list))
   yc-romaji-list
   (append (yc-subsequence yc-romaji-list 0 (+ yc-yomi-list-point dif))
   (cadr res) (nthcdr yc-yomi-list-point yc-romaji-list))
   yc-yomi-list-point (+ yc-yomi-list-point dif (length (car res)))))
     (setq yc-list-back yc-hiragana-list)))  ←この行を追加
```

この追加したコマンドは「yc-hiragana-list を yc-list-back に格納する。」という意味である。

格納したひらがなを出力するためには「変換を確定する関数」を改良する必要がある。yc-list-back から出力するには (insert (apply 'concat yc-list-back)) を追加すれば良い。以下がコマンドを追加したプログラムの一部である。

```
(defun yc-kakutei-internal ()
          :
          :
(yc-fence-mode nil) ; 表示していた文字を消去
(insert (apply 'concat yc-henkan-list))  ; 漢字を出力
(insert (apply 'concat yc-list-back))   ←この行を追加
(set-marker yc-fence-end (point))
          :
          :
```

このようにプログラムを変更した場合、yc-list-back からひらがなを出力することはできたがこれでは yc-hiragana-list から出力することが出来ないので問題点の解決にはならない。

5.2 変換を確定する関数の改良

yc-hiragana-list と yc-list-back の出力を条件によって分ければ良いと考え、プログラムを以下のように変更した。

```
(defun yc-kakutei-internal ()
          :
          :
(yc-fence-mode nil) ; 表示していた文字を消去
(insert (apply 'concat yc-henkan-list))  ; 漢字を出力

(if yc-list-back                                    ←この行から
    (insert (apply 'concat yc-list-back))             :
    (insert (apply 'concat yc-hiragana-list)))        :
(setq yc-list-back nil)                             ←この行まで追加

(set-marker yc-fence-end (point))
          :
          :
```

以下の追加したコマンドは 「yc-list-back に文字が格納されていれば yc-list-back から出力し、そうでなければ yc-hiragana-list から出力する」という意味である。

```
(if yc-list-back
    (insert (apply 'concat yc-list-back))
    (insert (apply 'concat yc-hiragana-list)))
```

その後、

```
(setq yc-list-back nil)
```

というコマンドで yc-list-back に nil を格納することで初期化している。

5.3 読みを入力する関数の改良

ひらがなを格納する場合も条件によって分けなければ常に yc-list-back に格納されてしまい残したいひらがなが消えてしまうので以下の

```
(if (not yc-list-back) (setq yc-list-back yc-hiragana-list))
```

というコマンドをどこかに追加する必要がある。

この上記のコマンドを「読みを入力する」関数に追加すると以下のようになる。

```
;; 読みを入力する
(defun yc-yomi-insert (str)
  (setq yc-fence-yomi (concat (substring yc-fence-yomi 0 yc-yomi-string-point)
      str
      (substring yc-fence-yomi yc-yomi-string-point))
                    :
                              :
                              :
  yc-romaji-list
  (append (yc-subsequence yc-romaji-list 0 (+ yc-yomi-list-point dif))
  (cadr res) (nthcdr yc-yomi-list-point yc-romaji-list))
  yc-yomi-list-point (+ yc-yomi-list-point dif (length (car res)))))
(if (not yc-list-back) (setq yc-list-back yc-hiragana-list)))  ←この
行を追加
```

この追加したコマンドは 「yc-list-back が nil の時 yc-hiragana-list を yc-list-back に格納する」という意味がある。

このようにプログラムを書き換えて実際に YC を動かしてみたところ、以下のように出力された。

- 海う
- 空 s

出力されるひらがながローマ字の最初の一文字しか出なかった。これは、yc-hiragana-list にひらがなが格納される時に、

- キーボードから u を入力 →う
- キーボードから m を入力 →う m
- キーボードから i を入力 →うみ

このような順番で格納されているため u を入力した時点で yc-list-back は nil でなくなったのでそれ以降の文字は格納されない。そのため、一文字目に入力した 「う」 のみが出力される。

この方法では正しくひらがなを格納できないので別に方法を考察する必要がある。

5.4 変換を取り消す関数の改良

「読みを入力する関数」ではうまくいかなかったので前章で考察した「変換を取り消す関数」yc-cancel の改良をする。yc-cancel なら入力した文字が全部 yc-hiragana-list に格納されている状態なので前節のような問題はおこらないと考えた。

yc-cancel に前節と同じコマンドを追加する。以下が追加したプログラムの一部である。

```
;; 変換を取り消す関数
;; 変換前の状態に戻す
(defun yc-cancel ()
  "漢字変換を中止し、変換前の状態に戻す"
  (interactive)
  (setq yc-mark-list (make-list (length yc-henkan-list) 0))
            :
            :
            :
(exit-minibuffer))))
(if (not yc-list-back) (setq yc-list-back yc-hiragana-list)))  ←この行を追加
```

yc-cancel の最後の行にコマンドを追加した。このようにプログラムを書き換えて YC を動かしてみたところ、以下のように出力された。

- 海うみ
- 空そら

また、「うみ」「そら」と入力した後、漢字変換モードから確定する前に C-g で削除しその後に、それぞれ「かい」「くう」と入力して確定した場合、以下のように出力された。

- 海うみ

- 空そら

これは、最初に入力した時 yc-list-back に格納されていたひらがなが出力されたということである。

これで「違う読みの入力」という問題点は解決したと言える。しかし、この方法にも問題点がある。

6 問題点

今回の研究で残された問題点を以下にまとめる。

- 一度入力した文字を別の文字に変える場合
  ひらがなを入力して漢字変換モードから C-g で全て消して次に全く違うひらがなを入力して変換し確定した場合、最初に入力したひらがなが出力されてしまう。

  > 例:「うみ」と入力→スペースキーで変換モードに移行→ C-g で削除→「やま」と入力→変換して確定→「山うみ」と出力されてしまう。

  これは yc-cancel を使うと起きる問題点なので変換モードに移行する前のひらがなを入力したところで C-g で削除すればこの問題は発生しないが、そうしなければこれはひらがなを残す機能としては問題である。

- 固有名詞など長い漢字を出力する場合
  固有名詞などを一度に入力して変換する時、一回の変換で正しい漢字が出力されなかった場合 C-g で削除して一文字ずつ変換する。その時、一文字目の漢字を出力した時最初に入力したフルネームのひらがなが出力されてしまう。

  > 例:「大桃隆宏」と表示させたい場合→「おおももたかひろ」と入力して変換→「尾主も高弘」と変換される→ C-g で削除→「おお」と入力して変換→「大おおももたかひろ」と出力されてしまう。

  しかし、この問題は「変換中の文節を伸ばす関数」yc-enlarge や「変換中の文節を縮める関数」yc-shrink を使って変換する文節を調整すればこの問題は考える必要がなくなる。yc-enlarge は C-o で yc-shrink は C-i でそれぞれ文節を調整できる。しかし、yc-enlarge や yc-shrink を使わなければこれは問題点になる。

7　まとめ

本研究はひらがなのみの文章を漢字かな混じりの文とともに残す機能の問題点の「違う読みの入力」という難しい漢字の読みを知らない時、または正しい漢字が変換しても出ない時に正しい読みのひらがなを残せないという問題点の解決を目的に研究をした。過去の卒業研究で追加されたコマンドはひらがなから漢字に変換された文字を確定で出力する部分に yc-hiragana-list という入力したひらがなを持つ変数の値を一緒に出力するものだった。yc-hiragana-list を調べたところ「入力したひらがなを格納する変数」であることがわかった。しかし、yc-hiragana-list では入力した文字を削除した時に yc-yomi-reset という関数が働いて正しい読みが残せない場合があった。そこで、ひらがなを格納しておく変数をもう一つ作ればこの問題は解決するのではないかと考え、新たに yc-list-back という変数を作って実験を行なった。

まず、yc-hirgana-list を yc-list-back にコピーし yc-list-back からひらがなを出力する実験を行なった。その結果 yc-list-back から出力することはできたが、この方法では yc-list-back からしか出力されないので問題の解決にはならなかった。

次に、「読みを入力する関数」と「変換を確定する関数」に条件付きのコマンドを追加して実験を行なった。その結果、入力した最初の一文字しか出力されなかった。

最後に、「変換を取り消す関数」に条件付きのコマンドを追加し実験した。その結果、yc-yomi-reset が働いても yc-list-back から出力することで正しいひらがなを出力することができた。

本研究で残された問題点をまとめる。一つ目が「一度入力した文字を別の文字に変える場合」である。ひらがなを入力して漢字変換モードから C-g で全て消して次に全く違うひらがなを入力して変換し確定した場合、最初に入力したひらがなが出力されてしまう。二つ目が「固有名詞など長い漢字を出力する場合」固有名詞などを一度に入力して変換する時、一回の変換で正しい漢字が出力されなかった場合 C-g で削除して一文字ずつ変換する。その時、一文字目の漢字を出力した時最初に入力したフルネームのひらがなが出力されてしまう。今回の研究ではこの二つの問題点が残ってしまった。また、過去の卒業研究で残されていた「わ」と読む「は」の問題点もひらがなを残す機能を作る上で解決しなくてはならない課題になっている。

参考文献

[1] 高橋慎二: 「Emacs で漢字の読みを残す機能の実装について」 新潟工科大学工学部情報電子工学科卒業論文 (2007)

[2] 広瀬雄二: やさしい Emacs-Lisp 講座 (株式会社カットシステム,1999)

[3] YC の部屋
http://www.ceres.dti.ne.jp/ knak/yc.html