Emacs で漢字の読みを残す機能の実装について

平成 19 年 2 月 3 日

情報電子工学科
高橋 慎二

目 次

1 はじめに **1**

2 かな漢字変換の基礎知識 **1**

2.1 Emacs (いーまっくす) について 1

2.2 自動変換をしたときの誤変換 1

3 かな漢字変換の構造 **3**

3.1 かな漢字変換をするソフトウェア 3

3.2 Canna (かんな) 3

3.3 かな漢字変換の仕組み 4

3.4 Emacs で Canna を使う方法 5

3.5 Emacs Lisp について 8

4 YC を使った実験 **10**

4.1 YC の構造 10

4.2 ひらがなを表示するための考察 12

5 ひらがなの出力形式 **15**

5.1 実験で出力されたひらがなについて 15

5.2 ひらがなを括弧で括る 15

5.3 セパレータ 17

5.4 次の行への出力 18

6 残った問題点 **19**

7 まとめ **20**

参考文献 **21**

概要

コンピュータで漢字の含まれる文章を作る場合は、まずひらがなを入力してから漢字に変換する。しかし、文章を点字に翻訳したり、音声として出力する場合、ひらがなが必要になるので、漢字からひらがなに戻すことになる。そのとき漢字からひらがなへの自動変換ソフトを使用すると漢字によっては誤変換が起こるので、その誤変換を避けるための方法として、文章を作るときに入力したひらがなが、漢字かな混じりの文章とともに残るようにすることを考えた。
本研究室では、主に Emacs というテキストエディタで文書を作成している。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。かな漢字変換も Emacs Lisp で作られているので、これを利用して、文章を作るときに入力したひらがなが残るような仕組みを考えた。Emacs Lisp で作られたかな漢字変換プログラムに YC があり、その YC を改良することでとりあえずひらがなを残すことができた。そして、漢字かな混じりの文章に対するひらがなを、どのような形で残せばいいかを考え、いくつかの方式を検討し、問題点をまとめた。

1　はじめに

文章を点字に翻訳したり、音声として出力する場合にひらがなの文章が必要になる。点字には漢字の点字は使われていない。そして、音声として出力する場合には、漢字毎に音をあてると、漢字の分だけ用意しなければならない。それより、ひらがなに直して各かなに音を割当た方が簡単である。以上のことから、ひらがなの文章が必要になってくる。

ひらがなを残すには、まずその仕組みから調べなければならない。仕組みを理解した上で、実際にひらがなを残すことができるか検討、考察していく。Emacs というテキストエディタは、Emacs–Lisp という言語による機能の拡張ができ、かな漢字変換もこれによって作られている。その Emacs-Lisp を利用して、漢字の含まれる文章を作るときにひらがなを残す機能を作ることが目標である。

2　かな漢字変換の基礎知識

2.1　Emacs (いーまっくす) について

最初に題目にある Emacs について説明する。

コンピュータで文字情報のみのファイル (テキストファイル) 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。

Emacs は長い歴史を持っており、ゼロからの書き直しを含む改良を重ね、多くの派生エディタを生んだが、オリジナルの EMACS は、1975 年にリチャード・ストールマンが、ガイ・スティール とともに作りあげた。

大文字で始まる「Emacs」と、小文字の「emacs」を区別する人もいる。大文字で始まる「Emacs」は、リチャード・ストールマンの作ったエディタから派生したエディタを指し、小文字の「emacs」は、たくさんある個別の emacs 実装を指す。

2.2　自動変換をしたときの誤変換

自動変換をすると漢字によっては誤変換が起こる。例として、本研究室にある kakasi というソフトを使って実験してみる。

kakasi (かかし) は、漢字かな混じり文をひらがなやカタカナに変換するソフトである。漢字の読めない端末を使った時や漢字に不慣れな外国人や子供に文章を紹介したい時などに使えるかもしれない。標準入力から日本語の文章を入力すると、指定された文字セットに変換されて出力される。

kakasi は、漢字かな混じり文をひらがなやローマ字に変換することを目的として、高橋裕信氏によって作成されたプログラムと辞書の総称である。kakasi という名前は、”kanji kana simple inverter” の略であり、日本語入力システムの SKK を逆に読んだものでもある。実際、kakasi で使用される辞書に含まれる単語の大半は SKK の辞書に由来するものである。

以下は、適当に選んだ WWW ページの文章を kakasi で変換した結果の誤りの部分である。

漢字	変換されたひらがな
中越	なかごし
流鏑馬	りゅうかぶらうま
坩堝	かんか
1 人	1 にん
第三次	だいさんつぎ
目される	めされる
時が経った	ときがへった
を行った	をいった
そんなある日	そんなあるにち
記憶を失っていた	きおくをうっていた
優れた武器	まされたぶき
魔術士	まじゅつさむらい
複数の敵を	ふくすうのかなを
その後は	そののちは

流鏑馬など難しい漢字の熟語や地名などの固有名詞などは、漢字からかなへ変換するための辞書にその言葉が無いのでうまく変換できない。これを改善するのは、難しくない。辞書にその言葉を追加すればいいのである。問題は 1 文字の漢字である。1 文字の漢字は読みがいくつかあるため正しく変換されにくい。「その後は」は、「そののちは」となっても、意味はわかるが、その他の誤りでは、意味がわからなくなっている。

また、「1 人」は 1 にんと変換されているが、「一人」と入力して変換すると、「ひとり」と正しく変換される。これは、半角の数字が、漢字と区別され、処理の対象になっていないからである。kakasi は、ある対象を、指定したもの (ひらがな等) に変換するが、ある対象とは、一種類なので、漢字と半角数字を同時に処理することができないために「1 人」のような文字はうまく変換されない。

なお、この誤変換は kakasi を使った結果であり、優れたソフトを使えば、誤変換は少なくなるが、完全に誤変換をなくせるわけではない。この問題を解決するには、送りがなや漢字の前の文字まで調べたり、文法を解析したり等、色々難しいことをしなければならない。しかし、すぐに誤変換の問題を解決することは、大変難しいので、漢字かな混じりの文章とともにひらがなの文章を残しておく必要がある。

3　かな漢字変換の構造

3.1　かな漢字変換をするソフトウェア

現在 UNIX でひらがなから漢字に変換するためのソフトウェアは数多くある。

例えば以下のものがある。

- Canna (かんな) 次節で説明。
- Sj3 (えすじぇーすりー) 非常に軽く、変換効率が高い。
- Wnn (うんぬ) 中国語や韓国語なども入力できる多言語入力システム。
- SKK (えすけいけい) かな漢字変換の変換アルゴリズムが特徴的。
- Anthy (あんしー) フリーでセキュアな日本語入力システム。
- ATOK (えいとっく) 歴史が長く、完成度や変換精度がよい。

その他、調べたものだけでも全部で 30 種類以上存在する。この研究では、本研究室で主に使われている Canna について調べる。

3.2　Canna (かんな)

Canna 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。'98 年に入ってから Windows 95 対応版も正式にリリースされている。

Canna の特徴を以下にまとめる。

- 押されたキーによって動作する内容を変更できる。
- クライアント・サーバ方式である。
- 複数の辞書を同時に使用することができる。

Canna は、実際に漢字変換を行う一つのサーバと利用者に対する日本語入力の処理と変換要求を行う複数のクライアントから構成される。この構成は、クライアント・サーバ方式と呼ばれ、Canna や Wnn 等でこの方式が使われている。

3.3 かな漢字変換の仕組み

かな漢字変換の仕組みについては以下の通りである。

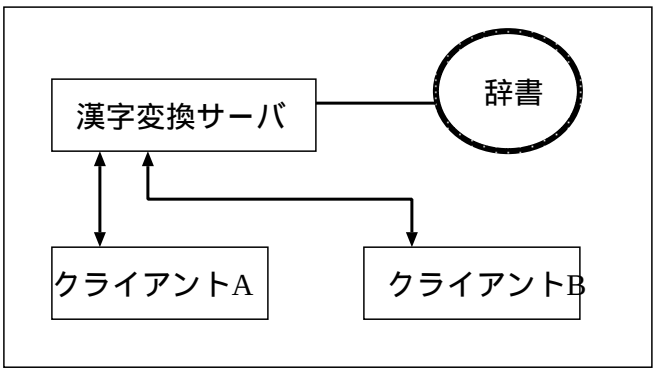

Fig. 1 クライアント・サーバ方式

- 漢字変換サーバ

  漢字変換サーバーには変換エンジン部分と辞書となる部分がある。クライアントから「ひらがな」などを受取り、漢字に変換して返す役割を持つもの。

- クライアント

  各ユーザーが共通のサーバを利用するための仕組みを担っているのが、クライアントである。ユーザーの入力を受け、漢字変換サーバと通信して漢字にしてアプリケーションに渡す役割をもつもの。ユーザーインタフェースを受け持ち、使い勝手に大きな影響を持っている。

  各サーバー付属の専用のものもあれば、別に開発され複数種の変換サーバーに対応しているクライアント (Wnn Canna Sj3 対応の kinput2 など) もある。

- アプリケーション

  Emacs はアプリケーション にあたる。アプリケーションによって対応している方式が異なり (クライアントと通信する方式が異なる) 。そのため、どの変換サーバーでも利用可能とは限らない。また変換クライアントを内蔵しているものもある。

クライアントは、インプットメソッドとも呼ばれる。世界には、文字数が多すぎてキーボードから直接入力できない言語がいくつかある。日本語もその中のひとつであり、そのような言語を入力するためのソフトウェアをインプットメソッドと呼ぶ。インプットメソッドには、二つの方式がある。

キーボードのキーをタイプすると、キーが押された事を知らせるためのキーイベントが発生する。発生したキーイベントがまずインプットメソッドに送られるか、それとも現在アクティブなアプリケーションへと送られるかで、インプットメソッドには 2 種類の形態が存在する。前者の場合のインプットメソッドをフロントエンド方式、後者の場合をバックエンド方式という。

- フロントエンド方式

  フロントエンド方式の場合はキーイベントはインプットメソッドが受けとり、アプリケーションに対しては変換が終わった文字列を送りつける。もちろん、インプットメソッドで必要としない場合にはキーイベントをアプリケーションに素通しする事もある。フロントエンド方式は単純であるが、アプリケーションと連携しない事が多いために使い勝手があまりよくない場合がある。

- バックエンド方式

  バックエンド方式の場合は、キーイベントを受けとったアプリケーションは、そこからさらにインプットメソッドに対してキーイベントを送りつける。インプットメソッドはそのキーイベントと現在の状態から現在の未確定文字列と確定文字列を計算し、アプリケーション側へと返す。未確定文字列はインプットメソッド側で描画する場合もある。

インプットメソッドという言葉はキーボードから直接入力できない文字を入力するためのソフトウェア、といった程度の意味合いで、変換エンジンもインプットメソッドの範疇に入るし、kinput2 などのようにアプリケーションと変換エンジンの間に入ってやりとりを行うソフトウェアもインプットメソッドに入る。

3.4 Emacs で Canna を使う方法

Emacs で Canna を使うには以下のいずれかの方法がある。

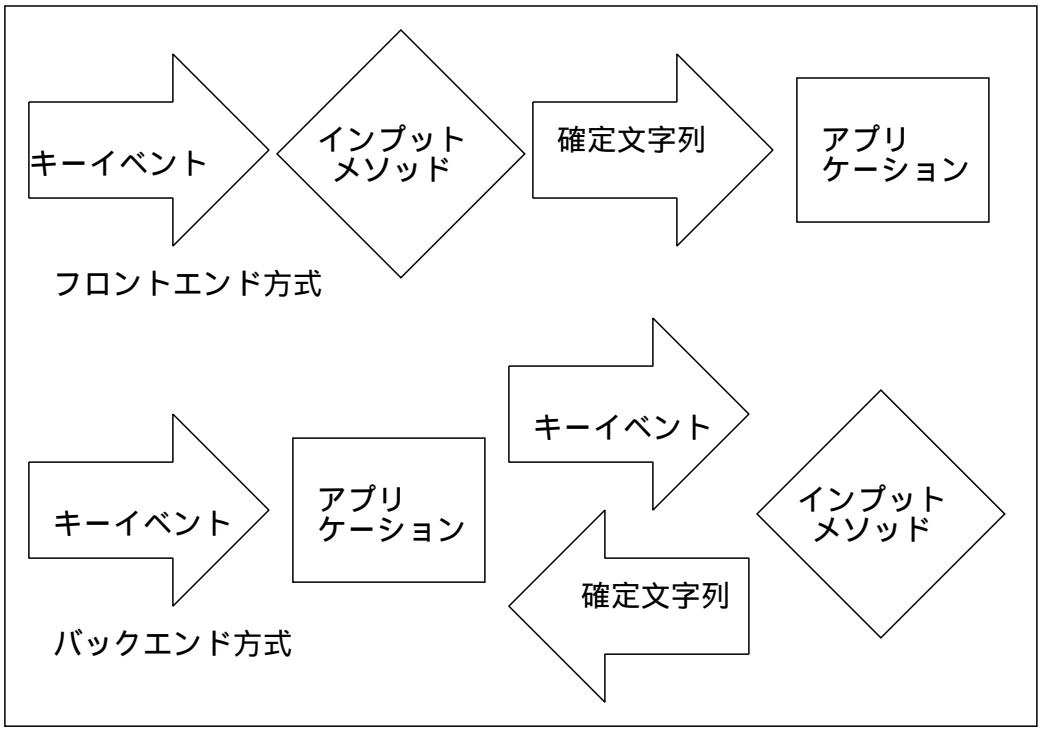

Fig. 2 フロントエンド方式とバックエンド方式

- Tamago 4 (+ egg-canna.el)

  Tamago 4 というものでは本来 Canna は使えないが、egg-canna.el によって、Canna を使うことができるようにしたもの。

- YC

  Emacs Lisp を使って Canna のサーバと直接通信できるようにしたもの。

- emcws

  Emacs に対するパッチファイル (Emacs 本体に対する修正部分) として提供されているもので、Emacs で Canna と Wnn と Sj3 を使うことができるようになる。

emcws については、インターネットを中心に調べたが、詳しいことは何もわからなかった。また、Tamago 4 (+ egg-canna.el) についても比較できるほど詳しく調べることはできなかった。

Tamago 4 の特徴について以下にまとめる。

- すべてが　Emacs Lisp だけで書かれている。

- 日本語と中国語の混じった文章を一括変換することもできる。
- emacs のバージョン 20.5 以降専用である

YC について以下にまとめる。

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

以下は、YC の長所である。

- Canna 対応していない Emacs でも Canna が使える。
- 半角アルファベットから直接漢字変換できる。
- 字種変換結果が候補に含まれる。
- 確定直後なら再変換できる。
- フェンスモードも使える。

以下は、YC の短所である。

- canna についての解析が不十分で、特に押されたキーに対する動作が Canna と違うものがある。
- canna の特定のバージョンにしか対応していない。

かな漢字変換をするサーバには、読みにあたるひらがなが送られていて、そのひらがなから変換される漢字の候補になるもの全てがクライアントへ返され、各クライアントやアプリケーションで一定の規則にしたがって表示される。

例えば、「きょう」という文字を入力して変換しようとすると、「きょう」というひらがながサーバに送られる。サーバは、「きょう」に対応する漢字をクライアントへ返す。その漢字は以下のものがある。「今日」、「教」、「共」、「強」等、登録されている漢字全て返す。クライアントは、最近使われた漢字を最初に表示するという規則があるなら、その規則にしたがって、「共」だけを表示して確定を待つ。目的の漢字でなければ、違う漢字の候補を表示して確定を待つか、または、候補になる漢字の一覧を表示し、選択、確定されるのを待つ。このようなやり取りをしている。

最近使われた漢字を最初に表示するという規則は、「今日」という漢字が、最近使われていれば、「今日」が最初の変換候補となる。

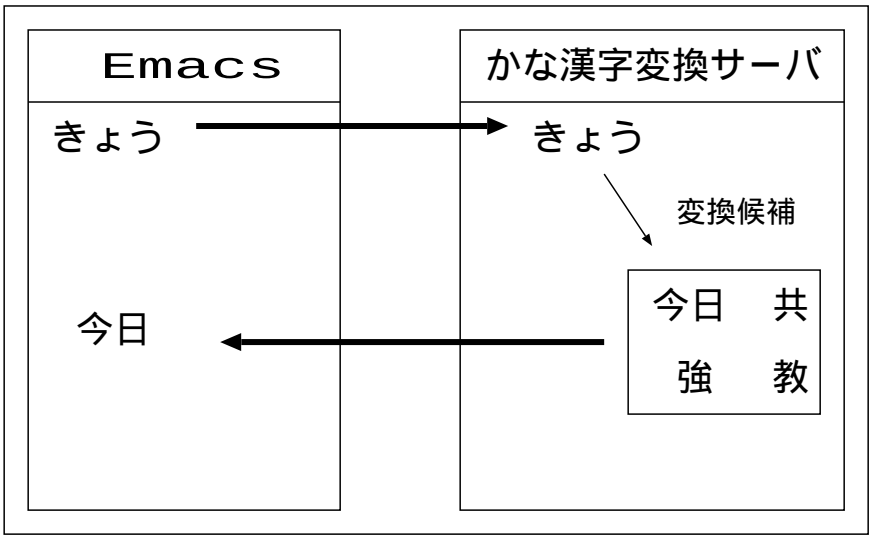

Fig. 3 漢字の変換例

ひらがながサーバに送られると、帰ってくるものは、漢字ばかりである。そのためサーバに送る前のひらがなを残しておくようにする方法を考える。

ひらがなを残すためには、バックエンド方式のものを扱うことにする。フロントエンド方式は、アプリケーションに対して、変換が終った後の文字列、つまり、漢字しか渡さないためアプリケーション上でひらがなを残すための処理ができないからである。

Canna を使う方法として上げたもので、YC がある。その YC の内容を変更して、ひらがなを残すことを考える。YC のプログラムは、emcws と比べて短い。YC は、約 3800 行に対し、emcws は、約 27000 行にもなる。そして、コメントが多く書かれているので変更すべき所がわかりやすく、全て Emacs Lisp で書かれている。以上のことから、簡単にできそうなので YC を選ぶことにする。

3.5 Emacs Lisp について

YC では、Emacs Lisp が使われている。Emacs Lisp とともに Lisp という言語について簡単に説明する。

Lisp (りすぷ、LISt Processing)

1962 年に MIT の John McCarthy 教授を中心とする研究グループによって開発された。代表的な関数型言語の一つで、既に定義されている関数を組み合わせて新しい関数を定義するという形でプログラムを記述する。人工知能研究などで利用されることが多い。

すべてのプログラミング言語を通じて Lisp は 2 番目に古い高級言語であり、現在でも広く使われている。Lisp の言語仕様は初期のころから大きく変化している。厳密には、Lisp は一つの言語ではなく「 Lisp 方言 」と呼ばれる同種の言語の集まりである。Lisp は実装の容易さゆえに非常に多くの方言を生んだ。以下はその系統と変種の一部である。

- LISP (純 LISP) MIT で開発されたマッカーシーのオリジナル版。
- Common Lisp Lisp の標準的仕様としてまとめられた。
- MACLisp MIT で作られた LISP の直系子孫。
- Scheme (すきーむ) 当初教育目的で設計された最小主義的な LISP。
- Emacs Lisp Emacs の拡張のための Lisp

その他、調べたものだけでも 10 種類以上がある。なにも付けずに「 Lisp 」というときは、一般的に Common Lisp のことを指し示している場合が多い。オリジナルの Lisp は、その言語仕様の簡潔さゆえに年月を経るうちに多くの方言が乱立し、相互の移植性が問題になっていた。この事態を解決する為、1980 年代に入って Lisp の統一仕様の策定が始まり 1994 年に至って第一版である標準仕様、情報技術プログラミング言語 Common Lisp が出版された。

以下に、 Lisp のコマンドの一部を紹介する。

```
(+ 1 2 3 4)
```

このように、記述する。これは、1,2,3,4 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。

```
(defun 関数名 ()
  "関数の簡単な説明"
  関数の内容)
```

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。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])
```

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。

```
(setq x 1)
```

setq というコマンドは、変数自体にに値を格納する関数である。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、処理が行なわれる。データ型は、関数、数値、リスト、配列などがある。

以下は、Emacs Lisp についての説明である。

Emacs Lisp とは Emacs が内蔵する拡張言語であり、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。

以下は、Emacs-Lisp 特有のコマンドである。

```
(insert [文字列])
```

現在、カーソルのある場所に、[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、複数でもできる。

4 YC を使った実験

4.1 YC の構造

YC のソースを見てわかったことを書く。YC はいくつかのモードにわかれている。

1. ひら仮名入力モード
   入力されたローマ字をひらがなに変換しながら入力するモード
2. ひら仮名編集モード
   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
3. 漢字変換モード
   指定された読みを漢字等に変換する

4. 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を紹介する。

まずは、ひら仮名入力モードのなかで使われている関数を紹介する。

関数名	コメント
yc-input-self-insert	入力されたローマ字をひら仮名に変換しながら読みを入力する

次に、ひら仮名編集モードで使われている関数を紹介する。

関数名	コメント
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 (追加入力や挿入等)
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	コメント
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 (変換前の状態に戻す)
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	コメント
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
yc-select-cancel	一覧モードを中止する関数

重要だと思われる関数について少し調べ、その結果を以下に示す。

- yc-edit-henkan

  ひら仮名編集モードで使われている関数である。変換のときに、ひらがながどのよう扱われているかを理解しなければならないので、この関数を調べる。調べた結果は、この関数の中でひらがなに対して、ほとんど処理をしていない。ひらがなは、入力されるときに変数に格納していて、その役目をしているのが yc-edit-self-insert という関数だった。yc-edit-henkan という関数は、ひら仮名編集モードから、漢字変換モードへ切り替える関数で、その後、yc-henkan-region という関数で、かな漢字変換をしていた。

- yc-henkan-region

  漢字変換モードで使われている関数である。かな漢字変換をするサーバにひらがなを送り、返ってきた変換候補となる漢字を変数に格納している。

- yc-kakutei

  漢字変換モードで使われている関数である。漢字変換を確定することは、ひらがなから漢字に変換するときに最後に行なうことである。この関数の動作によっては、ひらがなを残すために、この関数の内容を変更しなければならないと思われる。

4.2 ひらがなを表示するための考察

YC では、ひらがなを入力して変換キーを押すと、そのひらがなは漢字に変換され、漢字変換モードになる。目的の漢字でなければ、変換キーを押して別の候補を表示させる。そのようにして、一定回数変換キーを押すと、変換候補の漢字の一覧が表示される。その表示された一覧の中から選択するのが候補一覧モードである。

漢字変換モードか候補一覧モードで漢字を選択し確定すると、確定の処理をする関数 (yc-kakutei) が実行される。その確定の処理をする関数は、一度変換の候補等の表示に使った場所の文字を消して、漢字を変換した結果の文章を再び表示するような仕組みになっている。例えば、キーボードから kyou と入力して変換し、確定すると以下のようになる。

キーボードから k を入力	→	|k|
キーボードから y を入力	→	|ky|
キーボードから o を入力	→	|きょ|
キーボードから u を入力	→	|きょう|
変換キー（スペースキー）を押して変換	→	|今日|
変換キーを押して違う候補を表示	→	|共|
確定キー（リターンキー）を押すと一度消される	→	
確定の処理が終って結果を出力	→	共

確定キーを押したときに一度消されるが、普通は人の目には見えない。文字の両側についている ”|” は、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。また、変換キーを押すとフェンスで囲まれた文字が漢字に変換される。その他には、カタカナ等にも変換できる。フェンスで囲まれている状態をフェンスモードと呼び、リターンキー等で確定すると、フェンスモードが、解除される。

確定の処理をする関数の再表示する部分に、ひらがなも出力するように変更を加える。出力するひらがなは、変数に格納して残しておかなければならないが、 YC 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。この変数を使って変更を加える。以下は変更を加えたる前の確定を処理する関数である。

```
;; 変換を確定する関数
(defun yc-kakutei ()
  "漢字変換を確定する"
  (interactive)
  (yc-kakutei-internal))

(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))))
```

yc-kakutei という関数は、実は yc-kakutei-internal という関数を実行しているだけの関数である。

説明のために「; 表示していた文字を消去」と「; 漢字を出力」というコメントを書いた。「; 表示していた文字を消去」とコメントが書いてある行のコマンドは、(yc-fence-mode nil) となっているが、この部分は一つの関数で、yc-fence-mode という関数を使っている。この関数に引数として nil を指定すると、表示しているものを消去する働きがある。その次の行に「; 漢字を出力」とコメントで書いたが、その行のコマンドは、(insert (apply 'concat yc-henkan-list)) となっている。このコマンドが、確定した文字を出力する部分である。この後にひらがなを出力する命令を加える。yc-kakutei-internal という関数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
  (insert (apply 'concat yc-hiragana-list))    ; ひらがなを出力
          :
          :
)
```

確定の処理をする関数に変更を加えた結果、どのように出力されるか、例としてキーボードから kyou と入力して変換し、確定した結果を以下に示す。

キーボードから k を入力	→	|k|
キーボードから y を入力	→	|ky|
キーボードから o を入力	→	|きょ|
キーボードから u を入力	→	|きょう|
変換キーを押して変換	→	|今日|
確定キーを押す	→	今日きょう

5　ひらがなの出力形式

5.1　実験で出力されたひらがなについて

前節の方法でとりあえずひらがなを出力することができた。しかし、ただ隣に出力するだけでは、後で手作業で編集しなければならず大変である。そこで、どのような形でひらがなを出力すればいいか、考察する。

5.2　ひらがなを括弧で括る

前節のように変換した漢字と、その読みにあたるひらがながそのまま隣りに並んでいると区別しにくい。そのため区別することを考える。まずひらがなを括弧で括ってみる。

今日は [きょうは] 天気が悪い [てんきがわるい]

Emacs Lisp では、このように出力するのは、難しいことではない。例えば、確定のための関数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 "]")
           :
           :
 )
```

括弧で括ると、ひらがながそのまま隣りにあるよりは、見やすくなり編集しやすい。そして、このような形式の文章のファイルならば、C 言語 や perl という言語で処理をして、漢字かな混じりの文章、または、ひらがなだけの文章にすることもできそうであるが、括弧で括るだけでは問題がある。例えば以下のように出力されたとする。

今日は [きょうは] 天気 [てんき] がいい。外 [そと] に行こう [いこう]。

C 言語等で、漢字かな混じりの文章を取り出そうとすると、括弧と括弧の中の文字を消せばいいが、ひらがなの文章は、括弧の中の文字を取り出すだけではうまくできない。「きょうはてんきそといこう」となる。「天気」と「がいい。」を分けて入力するとこうなる。「がいい。」は、漢字に変換しないので、入力したらそのまま確定する。一度でも変換キーを押して漢字変換モードにならないと関数の yc-kakutei-internal が作動しないので、このようになる。そこで、括弧で括るだけでなく、以下のようにする。

[今日は|きょうは][天気|てんき] がいい。[外|そと] に [行こう|いこう]。

このようにすると、C 言語 や perl という言語で、漢字かな混じりの文章、または、ひらがなだけの文章を取り出しやすくなる。「[」と「 | 」、あるいは、「 | 」と「] 」の間にあるものに対して取り出し、「がいい。」のように外にあるものは、漢字かな混じりの文章、または、ひらがなだけの文章のどちらを取り出すときにも一緒に取り出せばいいのである。このように出力するには、関数 yc-kakutei-internal 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 "[")
  (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 "]")
           :
           :
)
```

しかし、まだ問題がある。括弧には様々な種類があるが、そのどれもが文章の中で使われる可能性がある。その括弧を対象に漢字かな混じりの文章、または、ひらがなだけの文章にする処理をすると、ひらがなだけの文章が欲しいのに、漢字が混ざった文章が出来てしまう。

5.3 セパレータ

ただ括弧で括るだけでは、問題がある。なので、その括弧を違うものにする。ただし、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。以下に例を示す。

```
  @@@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、セパレータ文字列と地の文が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るので、編集する必要があるときには、大変になる。また、処理をするプログラムも少し難しくなる。セパレータを出力するためには、確定のための関数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 "@@@KanjiPart@@@")
  (insert (apply 'concat yc-henkan-list))    ; 漢字を出力
  (insert "@@@KanaPart@@@")
  (insert (apply 'concat yc-hiragana-list))  ; ひらがなを出力
  (insert "@@@EndPart@@@")
           :
           :
 )
```

5.4　次の行への出力

漢字の隣りにひらがなを出力する方式を検討してきたが、ひらがなを隣りではなく、次の行に出力するようにしてはどうかと考える。例えば以下のようにする。

今日の天気は雨
きょうのてんきはあめ

ひらがなに対するセパレータのようなものが必要なら、次のようにする。

今日の天気は雨
@@@きょうのてんきはあめ

このような形式では、C 言語等で処理をして、必要な部分を取り出すのは、あまり難しくない。また、奇数行と偶数行に対して処理をする形式ならば、セパレータも必要無くなる。しかし、この形式でも問題はある。

ひらがなの読みがどの漢字に対応するのか分からないこと。例えば、漢字にふりがな (ルビ) を付ける場合に、ひらがなのどこからどこまでを漢字のふりがなとするかわかりにくいことである。ただし、これは目標とする機能ではない。括弧で括る方式ではできるが、この方式では、できないものである。

その他に、わかち書きの問題もある。わかち書きは、単語毎や文節毎に空白をいれることである。例えば、以下のようになる。

きょうの　てんきは　あめ

漢字かな混じりの文章は、単語等の区切りは、わかりやすいがひらがなやカタカナのみの文章では、区切りがわからない。そこで、わかち書きの必要があるが、ひらがなのみの文章からでは、わかち書きに直すことはできない。点字では、わかち書きされていないと意味がわかりずらく、音声では、どこで区切っていいのかわからないと意味がわからなくなってしまう。例えば、「こうしまるやさいいち」のようなものはわかち書きしなければ意味が違ってくる。「講師丸谷オー」と「こう閉まる野菜市」に読める可能性がある為である。

括弧で括る方式では、処理をするときに空白をいれればいいので、次の行へ出力する方式では、問題が多い。

6　残った問題点

- 違う読みの入力

  例えば、「流鏑馬 (やぶさめ)」という単語がある。この単語の読みを知らないとき、または、「やぶさめ」と入力しても変換候補として「流鏑馬」という漢字が出てこないときに、「りゅうかぶらうま」と入力して変換するかもしれない。あるいは、他の読み方で入力するかもしれない。そのときに入力したままのひらがなを残すと、意味がわからなくなってしまう。

- 半角文字の入力

  半角文字の入力は、通常かな漢字変換を利用せずに入力する。この研究では、一度フェンスモードにならないと、変換した漢字と別にひらがなを出力できない。例えば、日付を入力するときに、以下のようになる。

  ```
  今日の日付は 2007 年 1 月 19 日
  きょうのひづけはねんがつにち
  ```

  おそらく半角で入力するもの (数字やアルファベット) が、フェンスモードにならずに出力されるのでこのようになる。ただし、括弧で括って、漢字とひらがなを対応させる方法ならば、この問題は考える必要が無くなる。

- コピーの問題

  半角文字の入力と似たような問題であるが、ある文章をコピーして、もってきた場合に、その文章は、フェンスモードにならないので、漢字に対応するひらがなが残らない。半角文字の入力と違うのは、括弧で括る方式でもこの問題がある。確定するときの処理で、括弧で括った文字を出力するので、全て括弧の外に表示されてしまうので、うまく処理できない。

- 「わ」と読む「は」

  文章を音声として出力する場合には、「わ」と音読する「は」と普通の「は」は、区別できなければ都合が悪いが、全てひらがなであるとそれが「わ」の「は」か「は」の「は」か区別が難しい。漢字の隣で括弧で括ってあれば、前の文字との繋がりで判断できるかもしれない。しかし、完璧には出来ないと思われる。音声として出力するためにひらがなの文章が必要なので、「わ」と音読する「は」と普通の「は」は、区別できるようにしたい。

7　まとめ

本研究では、普段扱っている漢字かな混じりの文章ではなく、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ことを目的に研究した。始めに、かな漢字変換の構造を調べ、クライアント・サーバ方式であることがわかった。クライアント・サーバ方式は、クライアントとサーバにわかれていて、クライアントが、利用者の入力の処理とサーバに対する変換の要求をし、サーバは、要求にしたがってかな漢字変換をして、クライアントに結果を返すという働きをしている。クライアントは、インプットメソッドとも呼ばれ、サーバとアプリケーションをつなぐ役割もあり、そののつながりによって二つの方式がある。一方は、アプリケーションと連携しにくいため扱うことはなかった。そのため、もう一方の方式を対象に研究した。

YC は、インプットメソッドにあたるもので、Emacs (アプリケーション) の内部で動いているようで、ソースも Emacs Lisp で書かれていて研究には、都合がよかった。YC の構造について調べ、ひらがなを残すことができるとわかり、実際にソースを変更して実験をした。実験の結果、ひらがなは出力され、ひらがなを残すことができるとわかった。

ひらがなを残すことはできるので、どのような形式でひらがなが出力されると扱いやすいか考察した。ある方式にしたがってファイルを作り、そのファイルを C 言語等で処理をして、漢字かな混じりの文章、または、ひらがなのみの文章を取り出せるようにした。その方式は、漢字に対応するひらがなを括弧で括るなどの方式をいくつか考えたが、問題点がいくつか残ったので、それをまとめた。

この研究を始めたときから考えられた問題が、正しい読みが入力されないことである。正しい読みを知らないとき、または、正しい読みを入力しても、目的の漢字が変換候補に含まれないとき、違う読みを入力して変換する。そのひらがなが残るとひらがなの文章が必要なときに都合が悪い。そして、他の問題もひらがなを残す機能を作る上では解決しなくてはいけない課題になっている。

参考文献

[1] 小山 祐司, 斎藤 靖, 佐々木 浩, 中込 知之: UNIX 入門 フリーソフトウェアによる最新 UNIX 環境 (株式会社トッパン, 1996)

[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] ウィキペディア
http://ja.wikipedia.org/wiki/

[4] アスキーデジタル用語辞典
http://yougo.ascii24.com/

[5] 日本語入力 (canna) の使い方
http://www.ed.fuk.kindai.ac.jp/Tebiki/UNIX/Canna/

[6] 日本語入力システムかんなのページ
http://www.nec.co.jp/canna/

[7] 日本語入力に関する基礎知識
http://kodou.net/unixuser/200405/part1.html

[8] UNIX での日本語入力の仕組み
http://www4.airnet.ne.jp/koabe/com_inet/im/unix.html

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

[10] KAKASI 漢字→かな (ローマ字) 変換プログラム
http://kakasi.namazu.org/index.html.ja