T コード入力環境 tc2 マニュアル

Edition 2.4
2003 年 3 月

前田薫 maeda@src.ricoh.co.jp
斎藤靖 yasushi@cs.washington.edu
北嶋暁 kitajima@isc.osakac.ac.jp

1 T コードとは

『T コード』は東京大学理学部情報科学科山田研究室で開発された『無連想 2 ストローク漢字入力方式』です。ここでは、T コードについて、日本語の扱いが可能な Emacs(以降では単に Emacs と呼びます。) 用の T コード入力環境 tc2 を使用するにあたって必要最小限の紹介をします。

1.1 2 ストローク入力とは

『2 ストローク入力』はキー 2 打鍵の組み合わせで漢字 1 文字を表すことによって漢字入力を行う入力方式です。漢字入力は現在「かな漢字変換入力」が主流であると思われます。しかし、かな漢字変換は判断を必要とする作業であり、大量の文書を入力するのには向きません。一方、2 ストローク入力では、キーボードから直接漢字を入力できるので、変換の必要はありません。キーの 2 打鍵で常に一つの文字を入力することになるので、文章の入力はリズムをくずすことなく行えます。

> ■ただし、2 ストロークで表せる文字の種類には限りがあります。2 ストロークで直接入力できない文字のことを T コードでは『外字』と呼びます。外字の入力にはかな漢字変換を使うこともあります。Emacs 用の T コード入力環境 tc2 では、外字入力用に、部首合成変換 (4.2 節「部首合成変換」p.8 参照)・交ぜ書き変換 (4.3 節「交ぜ書き変換」p.11 参照)・JIS コード表 (第 4 章「T コードを使う」p.7 参照) などの補助入力手段を提供しています。□

漢字に対するキーの割り当てには『連想式』と『無連想式』があります。連想式というのはキーの組み合わせと文字の間に連想があるものを言います。たとえば「トキ」で「時」を表すなどの方法です。一方無連想式というのはそれらの間に連想関係がないものを言います。

一般に連想式は習熟にかかる時間は短くてすみますが、ある程度上達してしまうと打鍵速度が上がらなくなります。その理由は文字と打鍵位置の間に関係があるために入力が思考によって邪魔されてしまうためと、もともと入力する時のことは考えられていないために『awkward sequence』(複雑な指の運びを強いられる打鍵列のこと) が多くなり指に負担がかかってしまうためです。

無連想式では入力時の awkward sequence が小さくなるように打鍵位置を決められるため、習熟すればするだけいくらでも速く入力できるようになります。しかも変に思考に邪魔されることがないため、英文タイピストのようにおしゃべりをしながら文書の入力ができるようになるといわれています。そのかわり打鍵位置を覚えるまでは、何の手がかりもないのですから長い時間をかけてひたすら覚えることになります。

2 ストローク入力方式はたとえば手書きの文書を計算機に入力する時に最大の効果が期待できます。目から入ってきた文字を言語中枢を経由することなく手で打つことができるからです。しかし、頭で考えながら文章を入力するときには必ずしも最善の方法ではないかもしれません。

1.2 T コードで使用するキーボード

T コードでは片手あたり 5 列 4 段 20 個、計 40 個のキーを使います。40 個のキーを 2 打鍵することで 1 文字の入力を行います。人指し指は 2 列分を受け持ちます。T コードでは 40 個のキーの 2 打鍵、計 1600 通りの打鍵位置のうち現在約 1300 に文字を割り当てています。新聞などの記事を入力する際には、入力するべき文字の約 95–98%が T コードの二打鍵で入力できるといわれています。

普通のキーボードで T コードを利用する場合、次の図のような配列を用います。このマニュアルでは、いわゆる QWERTY 配列のキーボードを想定しています。以降の説明でも、特に断らない場合はこの配列に基づき

説明します。なお、T コード入力環境 tc2 では、Dvorak 配列用などにカスタマイズすることができます (5.6.2 節「入力文字と仮想鍵盤との対応」p.31 参照)。実際のキーボードでは、キーは下図のような長方形ではなく、一列毎にずれた配列になっていることが多いのですが、そこはがまんして使うことにします。

```
 左手用          右手用

1 2 3 4 5      6 7 8 9 0
Q W E R T      Y U I O P
A S D F G      H J K L ;
Z X C V B      N M , . /
```

■T コード専用のキーボードも存在します。T コード用のキーボードを見ると、その第一印象は真っ白なことです。真っ白なのはキートップに書くべき情報が何もないことから納得できます。次にもっとよく見るとキーが長方形に並んでいることに気づきます。キーの配置が長方形だというのは、「上段が左に、下段が右にずれて」いないということです。タイプライターに由来するこの「ずれ」は電子化された今のキーボードには必要ないからです。□

1.3 ストローク表の見方

T コードの打鍵位置の割り当てを表にまとめたものが『ストローク表』です。T コード入力環境上で 11, 10, 01, 00 を入力すればこの表を表示させることができます (第 4 章「T コードを使う」p.7 参照)。この表は大きく四つの部分に分かれています。それぞれ、5 × 4 の文字の長方形が 5 × 4 個並んでいます。そのうちの一つを見てみましょう。

RL

■■■■＊	■■■■■	■■■■■	■■■■■	■■■■■
請境系探象	尚賀岸責漁	舎喜幹丘糖	布苦圧恵固	姿絶密秘押
盛革突温捕	益援周域荒	康徒景処ぜ	邦舞雑漢緊	衆節杉肉除
依繊借須訳	織父枚乱香	譲へ模降走	激干彦均又	測血散笑弁
酸昼炭稲湯	貿捜異隣旧	攻焼闘奈夕	盤帯易速拡	汽換延雪互
歩回務島開	キせ区百木	や出タ手保	コ山者発立	ナ金マ和女
給員ど代レ	分よル千ア	７か（トれ	きっ日国二	上く８え年
相家的対歴	付プばュ作	内工八テ見	九名川機チ	サ建パ第入
桜瀬鳥催障	典博筋忠乳	採謡希仏察	君純副盟標	犯余堀肩療
中スもお定	わラ東生ろ	う４）十リ	あこ６学月	本さら高シ
３と〇てる	ーした一が	い、の５１	。＊０・２	ではになを
ッ人三京ち	ロク万方フ	んまンつ四	けイす電地	業時「長み
呼幅歓功盗	紀破郡抗幡	房績識属衣	去疑ぢ綿離	秒範核影麻
店持町所ほ	全じ自議明	バ部六経動	後間場ニ産	問ム七住北
行ド円小ジ	通カ社野同	だり一め大	新」９子五	事田会前そ
海道ず西げ	当理メウグ	不合面政オ	委化ビ目市	気売下都株

表の左上に 'RL' と書いてあるのは、この表が 1 打鍵目は右手、2 打鍵目は左手で打つ文字の表であることを示しています。小さなブロック内での文字の位置が 1 打鍵目の打鍵位置を、そのブロックが全体の中でどこにあるかが 2 打鍵目の打鍵位置を表しています。たとえば '円' という字は左下のブロックの中にあり、その中で

中央中段にありますから、1 打鍵目は右手中指のホームポジション、2 打鍵目は左手小指の下段 (Z のキー) ということになります。この表現形式を『木を見て森を見る方式』と評した人がいます。

この表の中で '■' で表された打鍵位置には現在文字が割り当てられていません。また、'*' で表された打鍵位置は、機能または 3 ストローク以上のキーストロークを表しており、文字は割り当てられていません。ここで、機能とは、たとえば部首合成変換入力 (4.2 節「部首合成変換」p.8 参照) の開始、交ぜ書き変換入力 (4.3 節「交ぜ書き変換」p.11 参照) の開始などを指します。また、3 ストローク以上のキーストロークは TUT コード (第 6 章「TUT コードで入力するには」p.36 参照) で使用します。

1.4 T コード情報の入手方法

WWW

T コードオープンラボ http://openlab.ring.gr.jp/tcode/で T コード関連の情報を公開しています。

Emacs 用の T コード入力環境 tc2 のサポートページは http://openlab.ring.gr.jp/tcode/tc2/です。

T コードメーリングリスト

T コード のメーリングリストが運営されています。T コードに興味を持つ人であれば、誰でも入会できます。内容、あるいは参加方法について知りたい人は、tcode-ml@is.s.u-tokyo.ac.jp 宛に# guide という内容のメールを送ってください。下はその例です。

```
% echo '# guide' | mail tcode-ml@is.s.u-tokyo.ac.jp
```

ニフティサーブからメーリングリストに参加したい場合には、INET:tcode-ml@is.s.u-tokyo.ac.jp あてに# guide の 1 行からなるメールを送ってください。

ニフティサーブ

ニフティサーブの FKBOARD では、T コード等の直接入力方式に関する会議室が開かれています。この会議室の最初の方には直接入力入門者のための非常に分かりやすい説明が多数登録されています。

また、次の Patio が直接入力に関するものです。

1. ID: NBH00475 (情報のやりとり)
   管理人: かぜはるか (email: nbh00475@nifterserve.or.jp)
2. ID: GGB03565 (関連データの登録)
   管理人: m(as)m (email: ggb03565@nifterserve.or.jp)

いちおうクローズドな形態になっていますので、参加を希望する方は、それぞれの管理人にメールして下さい。Password を教えてくれます。

2 個人環境の構築

ここでは、個人環境の構築方法について説明します。パッケージのインストールが正しく行われていることを前提にしています。パッケージのインストールの方法については、パッケージに付属の ‘INSTALL’ を見てください。

2.1 ‘.emacs’ での設定

まず、‘.emacs’ での設定を行います。

Emacs のバージョンが 20 以上か、XEmacs の場合は、次の記述が ‘.emacs’ にあるか確認して、なければ追加してください。

```
(set-language-environment "Japanese")
```

これにより、Emacs が、日本語文字コードで記述されたファイルの読み書きに適した環境になります。

インストールされた場所が Emacs に分かるように、load-path を設定することが必要な場合があります。‘*scratch*’ バッファで (require ’tc-setup) C-j と入力してみてください。エラーメッセージが表示された場合は、load-path の設定が必要です。‘tc.el’ などがどこにインストールされているか確かめて、次の記述を ‘.emacs’ に追加してください。

```
(setq load-path (cons "/path/to/tc" load-path))
```

上の例の中の */path/to/tc* の部分に、‘tc.el’ などがインストールされているディレクトリを記述します。

最後に、次の行を ‘.emacs’ に追加してください。

```
(require 'tc-setup)
```

ここまでで、‘.emacs’ での設定は完了です。設定を反映させるため、Emacs を再起動してください。

2.2 ‘.tc’ での設定

tc2 の設定は、設定ファイル ‘.tc’ に記述します。ここでは、その作成方法を示します。

Emacs を起動し、C-u C-\を入力します。そして、japanese-T-Code と入力します。ここで、TUT コードを使いたい場合は japanese-TUT-Code と入力します。すると、「設定ファイル~/.tc がありません。作成しますか?(y or n)」というメッセージが表示されます。ここで y と押すと、データを置くディレクトリとキー配置を聞かれますので、適切に入力します。

データを置くディレクトリには、辞書などのデータが置かれます。最初の状態は空ですが、必要に応じてファイルが置かれます。また、EELLL の練習テキストを自分用に作りたい場合などは、ここで指定したディレクトリに置いてください。なお、もとのデータは変数 tcode-site-data-directory で示されるディレクトリにあります。

キー配置については、使用するキー配置を選びます。大きく分けて、qwerty と dvorak があります。通常のキーボードをそのまま使用する場合は、qwerty を選択してください。また、入力時にシフトを用いるかどうかという選択肢があります。これは、キー配置名の次に-shift が付いているかいないかで区別できます。どちらにすればよいか分からない場合は、シフトは用いないでください。

上記の操作を行うと、‘~/.tc’ が作成されます。あとは、必要に応じて ‘.tc’ を編集してください (第 5 章「カスタマイズ」p.26 参照)。

3 T コードの練習

T コードで普通に入力できるようになるためには練習が必要です。実際の生活で使い始める前に、十分練習をすることが理想です。ただ、職業的に使用するのでなければ、ある程度修得したら、交ぜ書き変換などを用いて、そこそこ入力できるようにはなります。

練習には、EELLL を用いるとよいでしょう。

EELLL では、TUT コードの練習もできます。TUT コードの練習については、TUT-Code Home Page の、習得と練習テキスト (`http://www.crew.sfc.keio.ac.jp/~chk/shutoku.html`) も参照してください。TUT コード用の EELLL のテキストは、このウェブページにある練習テキストを採用しています。

3.1 T コード練習プログラム EELLL

『EELLL』は T コードや TUT コードの練習プログラムです。`EELLL` (「うなぎぎぎ」と読みます :-) は、山田研にかって存在したタイピング練習ソフト `Computer Aided Touch Type Trainer CATTT` (「きゃっとととと」) の Emacs Lisp 版であるということにちなんで名付けられました。端末を使って直接練習するプログラムとしては、`DOGGG` が用意されています (7.2 節「T コード練習プログラム DOGGG」p.37 参照)。

EELLL は `M-x eelll` で起動します。ミニバッファから練習テキストの番号を入力します。前回練習したレッスン番号がミニバッファにあらかじめ入力されています。基本的に、番号の順に練習を進めていってください。

練習テキストが 1 行ずつ出てきます。それを T コードを使って入力して下さい。入力はエコーバックされません。また、入力の際には部首合成変換や交ぜ書き変換を使うことはできません。

ビットマップを表示できる環境では、ビットマップによりヘルプが表示されます。うまく表示されない場合、変数 `eelll-use-image` を `t` に設定することで、ビットマップ表示されることもあります。

通常はテキストを順番に入力するだけですが、`M-x eelll-ramdom` で起動することにより、指定したレッスンの中からランダムに何行か選択して表示されます。1 回の練習で選択される行数は、変数 `eelll-random-max-line` で指定します。

EELLL に用意されているテキストを用いずに、練習する文を自分で作成したり何らかの文章を利用して練習したい場合、リージョンを指定して `M-x eelll-region` を実行すると、そのテキストの練習ができます。

3.2 練習のヒント

練習は基本的にはテキストの順に行えばよいのですが、ひらがなをだいたい覚えたら、数字やカタカナなどは後まわしにして、漢字に進んでも構いません。漢字が簡単に入力できることを早目に体験しておくと、練習意欲が増すかもしれません。

さて、練習にあたって心がけておくとよい点をあげておきましょう。

指の動きでコード (キーの入力順) を覚える。
: 例えば「kd」で「の」という覚え方ではなく、「右手中指→左手中指」の動きで「の」というように覚えましょう。

入力はリズムよく。
: 正確に打つことも大事ですが、一定のリズムで入力することがもっと重要です。EELLL では、1 行の入力が終わるまで入力した文字列が画面上に現れないようになっていますが、それはリズムを重視した練習を行いやすくするためです。

練習は継続的に。

毎日決まった量 (時間) を練習しましょう。なお、1 日に 2 時間以上練習しても効果は上がらないそうです。30 分とか 1 時間とか、時間を決めて効率的に練習しましょう。

覚えたはずの字のコードを忘れても気にしない。

人間誰でも忘れるもの。忘れたらまた覚えればいいだけのことです。

4 T コードを使う

4.1 使用法

C-\ (toggle-input-method がマップされているキー、 5.6.1 節「T コードモード切り替えキーの変更」p.30 参照) で、T コードを入力するモードに入ります。モード行が '[TC]' と変わります。

この状態で、たとえば jg と打つと、'は' が挿入されます。2 打鍵目にスペースを入力すると 1 打鍵目の文字がそのまま挿入されます。英小文字を少しだけ挿入したい場合に便利です。

シフトキーを押しながら入力すると、設定により動作が異なります。具体的には、キー配置名の設定で、末尾が '-shift' の配置を使うようにした場合、シフト機能が有効になります。シフト機能が有効な場合、デフォールトでは、ひらがなからカタカナへの変換が行われます。つまり、「の」を入力するためのストロークをシフトキーを押しながら行うと、「ノ」が入力されます。また、シフト機能が無効で、変数 tcode-shift-lowercase が t に設定されていた場合、その英字の小文字が直ちに挿入されます。たとえば、A と入力すると、'a' が挿入されます。シフト機能が無効で、tcode-shift-lowercase が nil の場合、入力した文字がそのまま入力されます。シフトを有効にしたい場合は、'.tc' で tcode-set-key-layout の配置の部分を、'-shift' の付いている分に変更してください。どのような配置があるのかは、変数 tcode-key-layout-list の値を参照してください。

1 打鍵目を打った直後にその 1 打鍵目を取り消すには、DEL(バックスペース) を入力します。

Emacs のコマンド引数も使用できます。文字を挿入するストロークの前に C-u と入力すれば、次に入力した文字が二つ挿入されます (挿入される文字の数が非 T コードモードの場合と異なります)。整数を指定した場合 (C-u 3 など) はその数だけ挿入されます。

もう一度 C-\を打つと、非 T コードモードに戻ります。

ミニバッファでも、C-\を入力することで、T コードを使うことができます。ミニバッファで T コードモードになっているときは、ミニバッファの左端に '[TC]' と表示されます。

T コードモードでは、2 ストロークで漢字を入力する以外に、いろいろな機能が用意されています。これらの機能は、T コードコード表の空の所に割り当てられています。次に列挙します。

jf　前置型の部首合成変換を開始します (4.2 節「部首合成変換」p.8 参照)。
ただし、変数 tcode-use-postfix-bushu-as-default を t にすると、77 と jf の機能が入れ換わります。

fj　交ぜ書き変換を行います (4.3 節「交ぜ書き変換」p.11 参照)。

55　現在のカーソル位置にある文字の打ち方 (ヘルプ表) を表示します (4.6 節「様々なヘルプ機能」p.21 参照)。

44　ヘルプ表で表示された打ち方で最も新しい分を再度 55 と同様に表示します。

11　'LL' (第 1 ストローク:左、第 2 ストローク:左) のストローク表 (1.3 節「ストローク表の見方」p.2 参照) を表示します。なお、ストローク表を表示するためのキーストロークは、1 が左、0 が右に対応しています (最上段の、T コードで用いるキーの左端と右端)。

10　'LR' (第 1 ストローク:左、第 2 ストローク:右) のストローク表を表示します。

01 ‘RL’ (第 1 ストローク:右、第 2 ストローク:左) のストローク表を表示します。

00 ‘RR’ (第 1 ストローク:右、第 2 ストローク:右) のストローク表を表示します。

32 句読点セットを「、。」と「, . 」の間で切り替えます。デフォールトは「、。」です。変数 tcode-switch-table-list を使うと、自分の好みの句読点セットを利用することもできます (5.7 節「個人用設定ファイルの見本」p.33 参照)。

77 カーソル直前の 2 文字を部首合成変換によって合成した文字で置きかえます (4.2 節「部首合成変換」p.8 参照)。jf が前置型の部首合成変換であるのに対して、77 は後置型の部首合成変換です。

88 現在のカーソル位置にある文字を、1 打鍵目と 2 打鍵目を入れかえて打ったときの文字で置きかえます。たとえば、‘味’ の上にカーソルを置いて 88 を打てば、‘味’ が ‘の’ に変わります。行末にカーソルがある場合には、その直前の文字に対してこの処理を行います。

99 現在実行途中の部首合成変換・交ぜ書き変換を中断します。また、ヘルプ用のウインドウを消します。

22 別バッファ(*jis-code*) に JIS の全文字をコード順に表示します。このバッファ上で適当な文字の上にカーソルを合わせて RET を押すと、その文字が、直前にいたバッファに挿入されます。q を押すと、直前にいたバッファに戻ります。

33 英数字の文字コードを 2 バイト・1 バイトの間で切り替えます。

58 活用語を優先して交ぜ書き変換を行います (4.3.2.3 節「活用する語を変換する際の注意点」p.14 参照)。

[1-4]8, [2-5]9
文字数を指定して後置型交ぜ書き変換を行います (4.3.2.4 節「文字数を指定した交ぜ書き変換」p.15 参照)。

なお、T コードモードで?を押すと、以上の、キーと機能との関係の要約が表示されます。

4.2 部首合成変換

外字を入力するための補助入力機構の一つに部首合成変換があります。

4.2.1 部首合成変換とは

外字の入力をするのに、T コードに割り当てられている文字の部首を組み合わせる方法があります。これを『部首合成変換』と呼んでいます。

T コードの部首合成変換で使用する部首は、実際には漢和辞典にのっている部首ほど正確ではなく、文字の形を元にかなりいいかげんにできています。以下に主な部首を代表する文字を示します。

ア、院→こざとへん	イ→にんべん	ウ→うかんむり
エ→工	オ→てへん	サ→くさがんむり
シ→さんずい	ヌ→又	ネ→しめすへん
リ→りっとう	レ→礼のつくり	ロ→口 (くち)
ワ→わかんむり	ン→にすい	部→おおざと
性→りっしんべん	独→けものへん	四→あみがしら
図→くにがまえ	之→しんにょう	

これらの文字だけでなく、部首を構成要素として含んでいる文字で代用することもできます。
以下に、部首合成変換のいくつかの例を示します。

ロ + ル→兄　　日 + 生→星　　シ + 談→淡　　点 + 重→薫
病 + 波→疲　　図 + 木→困　　国 + 木→梱　　頭 + 川→順
石 + 白→碧　　言 + 売→読　　ア + 良→限　　水 + う→永

また、部首の引き算もできます。

頭 − 豆→頁　　例 − イ→列　　列 − リ→歹　　麻 − 木→床
（頭 = 豆 + 頁）　（例 = イ + 列）　（列 = 歹 + リ）　（麻 = 床 + 木）

また、共通の部首を取り出す場合もあります。

題 + 題→頁　　進 + 進→之

いくつかの記号は、意味を表す文字で入力します。

た + す→＋　　ひ + く→ー　　か + け→×　　わ + る→÷
等 + 号→＝　　不 + 等→≠　　矢 + 上→↑　　、 + 、→,

ただし、記号はデフォールトの辞書には登録されていません。この変換を行いたい場合は、辞書に登録してください (4.2.4 節「部首合成変換辞書」p.11 参照)。
2 文字の組み合わせで入力できない文字は、3 文字以上の組み合わせで入力できる場合があります。

劇 =（七 + 上→虎）+ リ

まれに、組み合わされる 2 文字の順序が関係ある場合があります。

門 + オ→閉　　オ + 門→捫　　足 + 戔→践　　戔 + 足→踐

部首合成変換は部首合成変換辞書に基づき行われます (4.2.4 節「部首合成変換辞書」p.11 参照)。tc2 での部首合成変換アルゴリズムはちょっと変わっています。実際にどの文字が入力されるのかは、一般に、組み合わせる 2 文字を打鍵してみるまでわかりません。

4.2.2 tc2 での部首合成変換

T コードモードで jf と打つと部首合成変換モードに入ります。画面には '▲' と表示されます。この状態で、T コード文字を二つ入力すると、その 2 文字を合成してできる文字にそれらが置きかえられます。合成してできる文字がないと、beep 音がなります。
部首合成変換は再帰的に使うこともできます。たとえば、'劇 [(七+上→虎)+リ]' という文字を入力したい時には、

キー		画面
jf	（部首合成変換に入る）	▲
jf	（さらに部首合成変換に入る）	▲▲
ib	（七）	▲▲七
ht	（上）	▲虎
pd	（リ）	劇

のようにして入力することができます。

また、T コードモードで 77 と打つと、カーソルの直前の 2 文字を合成して、その 2 文字を合成結果の文字に置きかえます (後置型部首合成変換)。こちらの方が分かりやすいかもしれません。

キー		画面
ib	(七)	七
ht	(上)	七上
77	(合成)	虍
pd	(リ)	虍リ
77	(合成)	劇

■あまりに複雑な合成が必要な文字については交ぜ書き変換を使う方がよいでしょう (4.3 節「交ぜ書き変換」p.11 参照)。□

部首合成変換辞書の初期化 (辞書の読み出し) は、処理能力の低い計算機では時間がかかることもあります。このため、必要となるまで初期化を行わないようにしたいと思うかもしれません。Emacs の変数 tcode-bushu-on-demand を 0 以外の値に設定すると、その値に応じて辞書初期化のタイミングが変わります。5.4 節「ユーザ変数」p.27 を参照してください。

なお、辞書の初期化がされないうちは部首合成変換は使用できません。

4.2.3 対話的な部首合成変換

コマンド tcode-bushu-convert-preceding-char-interactively を使えば、部首を順次入力して、インクリメンタルに文字を選択することができます。このコマンドは、デフォールトではキーに割り当てられていません。使いたい場合は、次のように設定してください。

```
(add-hook 'tcode-ready-hook
  (function
   (lambda ()
     (tcode-set-action-to-table '(5 5) ; 66 で対話的な部首変換
'tcode-bushu-convert-preceding-char-interactively))))
```

このコマンドは、入力された 1 文字に対し、対話的に変換候補を表示します。例えば、「山」と入力し、変換すると、エコーエリアに次のように表示されます。

山 => 岡 [岳岩岸端缶嵩仙岨迚……

この例では、'=>' の左側にある '山' の部分が現在の対象部首の集合で、'=>' の右側にある '岡' が変換候補です。'[' 以降が他の候補を表しています。この状態で使用できるキーは次のとおりです。

SPC

> 変換候補を次の字へ移す。

< 変換候補を前の字へ移す。

RET 先頭の候補を確定する。

文字　　文字を対象部首に追加して変換候補を絞る。

Backspace　対象部首を 1 文字し、一段階前の状態に戻る。対象部首がなくなる場合は変換自体を中断する。

4.2.4 部首合成変換辞書

部首合成変換辞書 'bushu.rev' には、1 行に 1 文字分のエントリがあります。

　　唖口亜

上の行は、'唖' という見出しの文字は '口' と '亜' の 2 文字から成ることを意味します。現在の辞書では、過去の経緯から、見出しの文字を構成している文字は 1 文字または 2 文字で構成されていますが、何文字でも構いません。

　　院ア

上のように、1 文字で構成されている場合、その文字と見出しの文字は等価である、ということを定義しています。つまり、'院' はこざとへんを代表する文字ですが、'院' の代わりに 'ア' を使っても、こざとへんを入力できるのです。ここで、'院' と 'ア' の順番に注意してください。見出しの文字の方が代表する文字です。

部首合成変換辞書への登録や削除は 'bushu.rev' を編集することにより行います。ここで、辞書は EUC コードでソートされていなければなりません。登録したときは、忘れずにソートしてください。

標準辞書 'bushu.rev' とは別に、記号類を登録してある 'symbol.rev' があります。この辞書を使いたい場合は、その内容を 'bushu.rev' に追加して、ソートしてください。Emacs を用いてソートする場合は、'bushu.rev' に C-x i などで 'symbol.rev' 追加して、C-x h M-x sort-lines の後 'bushu.rev' を保存してください。

4.3 交ぜ書き変換

T コードで外字を入力するには、部首合成変換のほかに交ぜ書き変換という方法もあります。

4.3.1 交ぜ書き変換とは

T コードを使って熟語変換のようなことをすることも考えられます。この場合、T コードで入力できる漢字を直接入力することで同音異義語が減り、通常のかな漢字変換よりも効率のよい変換をすることができます。

　　き社のき者がき車でき社した
　　き社→帰社、貴社　き者→記者　き車→汽車

このような変換を、読みとして漢字とひらがなを交ぜ書きして指定できることから、『交ぜ書き変換』と呼びます[1]。

交ぜ書き変換では、接頭語・接尾語は T コードで入力できることが多いのでいちいち接頭語つきの複合語として辞書に登録する必要はありません。そのかわり、一つの単語について「読み」の組み合わせが増えるため、辞書が大きくなってしまいます。

[1] 読みをひらがなだけで指定した場合でも、ここでは交ぜ書き変換と読んでいます。「交ぜ書きが可能な変換」というくらいの意味です。

郵便　ゆうびん、ゆう便、郵びん
方程式　ほうていしき、方ていしき、ほう程しき、ほうてい式、
方程しき、方てい式、ほう程式

一方、直接入力できる漢字のみからなる候補は、辞書に登録されていてもまず使用しないので、削除した方がよいかもしれません。その方が、同音異義語が減り、選択がしやすくなるからです。 4.3.7.2 節「交ぜ書き変換辞書の作り方」p.18 を参照してください。 また、 4.3.5.2 節「交ぜ書き変換辞書からの削除」p.16 を参照してください。

4.3.2 tc2 での交ぜ書き変換—後置型

tc2 標準の交ぜ書き変換は、この「後置型」です。

この後置型交ぜ書き変換では、変換の前に特にモードに入ることなく適当に読みを入力して、最後に fj と打つと、カーソル直前の文字列の中で辞書に登録されている最も長い読みを使って (いわゆる最長一致) 漢字変換が行われます。

4.3.2.1 候補の選択法

候補の選択法は、入力した読みに対する辞書中の候補の数によって異なります。

読みに対し、候補が一つの場合
: その候補に変換されますので、その候補を選択するためには、次の入力を始めるか、または RET により明示的にその候補を確定します。

  なお、前置型の変換 (4.3.3 節「tc2 での交ぜ書き変換—前置型」p.15 参照) で、活用する語の変換をしないように設定している場合 (5.4 節「ユーザ変数」p.27 参照) は、自動的に確定します。

候補がちょうど二つの場合
: たとえば、読み「かん定」に対する漢字として、「勘定」「鑑定」の二つだけが辞書に登録されているとします。このとき、「かん定 fj」(後置型の場合) と入力すると、次のように候補が表示されます。

  △{勘定,鑑定}

  ここで左側に表示されている候補を選ぶには、第 1 候補を選択するキー (デフォールトでは d) を押します。右側の候補を選ぶには、同様に、第 2 候補を選択するキー (デフォールトでは f) を押します。

  選択に用いるキーは、変数 tcode-mazegaki-alternative-select-first-keys と変数 tcode-mazegaki-alternative-select-second-keys の値を変えることで変更できます。いずれかの変数の値が nil の場合は、この選択法は用いず、候補が 10 個までの場合と同じになります。

候補が 10 個までの場合
: たとえば、「あわ fj」(後置型の場合) と入力すると、ミニバッファに一覧が次のように表示されます。

  [- 泡 阿波 粟] - - [- - - -]

これは、T コードで使用するキーの、上から 3 段目を表しています。候補を選択するときは、選びたい候補に対応しているキーを押します。上の例では、「泡」を選択したいときは、T コードで使用するキーの、上から 3 段目、左から 2 番目のキー (S のキー) を押します。

候補が 10 個を超える場合

たとえば、「こう fj」と入力すると、ウインドウに次のように表示されます。

```
 -    -    -    -    -       -    -    -    -    -
[孔   幸   厚   黄  ] 逅     鈎 [ 肛   浩   袷   恰]
[頁   煌   候   稿  ] 項     耕 [ 膠   淆   功   侯]
[-    -    -    -   ] -       - [  -    -    -    -]     (1/5)
```

この表は T コードで使用するキーを表しています。候補を選択するときは、選びたい候補に対応しているキーを押します。なお、候補の優先度と入力するキーとの対応は、変数 tcode-mazegaki-stroke-priority-list で指定できます (5.4 節「ユーザ変数」p.27 参照)。

表の右端の '(1/5)' は、候補の数が表三つ分あり、その表のうちの 1 番目を表示していることを示しています。表を切り替えるには、次の表に切り替える場合は SPC を、前の表に切り替える場合は DEL を押します。

確定前の状態 (交ぜ書き変換候補選択モード) で利用可能なキーは次のとおりです。

SPC　次の表または候補を表示する。

DEL　前の表を表示する。

RET　候補を確定して、候補選択モードを抜ける。

<　読みを伸ばす。

>　読みを縮める。

C-u　候補を読みの状態にもどして、候補選択モードを抜ける。

|　漢字を登録して、その後変換する。

!　現在の読みの漢字を (一つ選んで) 削除する。

C-b　現在の区切りの先頭部分にカーソルを移動して、そこから後置型変換を行う (読みをさかのぼって変換する)。確定したら、その前の候補選択に戻る。

C-f　さかのぼった変換を一つ分取り消す。

上に挙げた以外のキーを入力すると、現在の候補を確定して、候補選択モードを抜けた後、入力されたキーを再実行します。

候補の先頭 4 個を超えた候補を確定すると、その候補は 5 番目の候補になり、次回の出現順 (一覧表での並び方) が変わります (先頭 4 個の候補の出現順には影響を与えません)。一方、候補の先頭 4 個までを確定したときは、次回同じ読みを変換しても、その出現順は変わりません。候補入れ替えの対象とならない候補の数は、変数 tcode-mazegaki-fixed-priority-count で指定できます (5.4 節「ユーザ変数」p.27 参照)。

4.3.2.2 活用する語に対する変換候補の検索法

交ぜ書き変換モジュールでは、活用する語に対する変換候補の検索は、文法の情報を用いず、パターンマッチングのみにより行います。辞書には、読みとして、たとえば「ながめ—」のように、語幹 'ながめ' と活用する語であることの印 '—' をつなげたものを登録します。

パターンマッチングによる変換は次のように行われます。たとえば、「ながめて」と入力して変換した場合には、「ながめ」と「て」とに分割すれば「ながめ」が辞書の読みの「ながめ」と一致しますので、「ながめ」の部分を「眺め」にし、語尾「て」をつなげたものに変換します。

パターンマッチングのみによる変換の問題点としては、文法に合わない語尾でも変換候補になることが挙げられます。この問題点に対し、交ぜ書き変換モジュールでは、「人が選ぶので、おかしな候補があってもよい」という立場で、これ以上の自動的な候補削減を行わないことにしています。

逆に、パターンマッチングのみによる変換の長所は、古文や方言に柔軟に対応できることです。

4.3.2.3 活用する語を変換する際の注意点

活用する語は、以下の各項目に注意すれば、活用しない語と同様に変換できます。

- 活用しない語と活用する語は、異なる読みとして扱われます。

  たとえば、「葵」と「蒼い」は、両方とも読み方は「あおい」ですが、「葵」は活用しない語、「蒼い」は活用する語です。ですから、変換の際に表示される読みの一覧の中に、これらの候補が同時に現れることはありません (「蒼い」が活用しない語として登録されている場合を除く)。

- 活用しない語は、通常、活用する語よりも優先されます。

  上の「あおい」の例では、まず「葵」が表示され、読みを縮める (> を入力する) と、活用しない語として読みを縮めて区切り直します。つまり、この場合では、読みを縮めていくと、「おい」、「い」の順に活用しない語の候補一覧が (もしあれば) 表示されます。その後、更に読みを縮める[2] と、読みを、活用する語として区切り直し、「蒼い」が候補となります。

  活用する語であることを明示して変換すれば、活用する語である「蒼い」をすぐに選択できるようになります。これは、`fj` の代わりに `58` (または `C-u fj`) を入力することにより行えます。

- 活用する語の区切り直しは、語幹の長さに基づいて行われます。

  活用する語は『語幹』と『語尾』に分かれますが、区切り直しは、縮める方向の場合では、語幹の長い候補が優先され、また、その中で語尾の長い候補が優先されます。

- 活用語尾にはひらがな・カタカナしか使えません。

  たとえば、読み「お—」・漢字「押」が登録されていても、「押し通す」を入力しようとして、「おし通す」という読みで変換することはできません。この場合は、「おし」という読みで変換し、その後、「通す」を入力してください (または、「おし通—」という読みを登録してください)。

[2] 『縮める』という語を使っていますが、この例のように、実際には長くなることもあります。

4.3.2.4 文字数を指定した交ぜ書き変換

以下のキーストロークにより、後置型で変換対象 (読み) の文字数を指定して交ぜ書き変換ができます。

活用しない語

キーストローク	読みの長さ
18	1
28	2
38	3
48	4

活用する語

キーストローク	読みの長さ
29	2
39	3
49	4
59	5

ここで、活用する語の読みの長さには、活用語尾の長さも含みます。

また、fj のコマンド引数として文字数を指定することができます。C-u fj の場合は活用する語 (長さは自由)、正の整数 (たとえば C-u 3 fj) の場合は活用しない語で文字数を指定、負の整数 (たとえば C-u - 3 fj) の場合は活用する語で文字数を指定、というようになっています。

文字数を指定した場合、活用しない語では、指定した読みに対し候補が一つしかない場合は、変換後ただちに確定します。

4.3.3 tc2 での交ぜ書き変換—前置型

変数 tcode-use-prefix-mazegaki を t にすると、前置型の交ぜ書き変換を使うことができます。T コードモードで fj と打つと『交ぜ書き入力モード』に入ります。このモードから抜けるときは、99 を押します。このモードで読みを入力し、SPC を押すと、交ぜ書き変換候補選択モードに入ります。候補選択の方法は後置型の場合と同じです (4.3.2.1 節「候補の選択法」p.12 参照)。

4.3.4 読みの補完

交ぜ書き変換辞書の読みから、入力途中の文字列を補完する (未入力の部分を補う) ことができます。「読み」といっても、交ぜ書きですから、すべて漢字でも構いませんし、カタカナでも英単語でも構いません。

使用法を例で説明します。ここでは、辞書に読み「アルゴリズム」、漢字「アルゴリズム」が登録してあるとします。

「アル」と入力し、=を押すと、「アル」で始まる読みが他にない場合では、「アルゴリズム」と補完され、その読みで変換されます。また、「アルゴリズム」の他に「アルゴン」という読みがあった場合では、「アルゴ」まで補完され、ミニバッファに

```
アルゴ{リズム, ン}
```

と表示されます。'{}' の中が候補です。この候補に対し、次の操作ができます。

SPC　次の候補を先頭へ

DEL　最後の候補を先頭へ

=

TAB

LFD　先頭の候補を選択し、それを読みとして交ぜ書き変換

RET　補完をそこまでで中断

4.3.5 交ぜ書き変換辞書の管理

交ぜ書き変換辞書への漢字の登録および辞書からの削除の方法を説明します。

4.3.5.1 交ぜ書き変換辞書への登録

交ぜ書き変換辞書への登録の方法には、一つの漢字を登録する場合と、複数の漢字をまとめて登録する場合の 2 通りあります。

一つの漢字を登録する場合

T コードモードで|を押します。その後、読み・漢字を順に入力していけば登録できます。

ここで、交ぜ書き変換の場合、同じ漢字に対して複数の読みを登録したい場合があります。たとえば、「春夏秋冬」という漢字に対して、「春」だけ入力法を知っている場合には、「春夏秋とう」「春夏しゅう冬」「春夏しゅうとう」「春か秋冬」「春か秋とう」「春かしゅう冬」「春かしゅうとう」という 7 通りの読みがあります。これは、読みを「春|か|しゅう|とう」、漢字を「春|夏|秋|冬」と入力することで、一度に登録できます。

複数の漢字を登録する場合

まず、次の形式のデータを用意します。

読み *1* **/漢字** *1-1*/**漢字** *1-2*/. . ./**漢字** *1-m*/
:
読み *n* **/漢字** *n-1*/**漢字** *n-2*/. . ./**漢字** *n-l*/

次に、このデータがあるバッファ上で M-x tcode-mazegaki-make-entries-buffer とします。なお、すでに登録されている読みと漢字との組み合わせについては、二重登録はしないようになっています。

4.3.5.2 交ぜ書き変換辞書からの削除

交ぜ書き変換辞書からの削除の方法には、一つの漢字を削除する場合と、複数の漢字をまとめて削除する場合、入力法を覚えた漢字一字について不要な漢字をまとめて削除する場合の 3 通りあります。

一つの漢字を削除する場合

T コードモードで C-u !を押し、読みを入力します。変換中や変換中断直後なら!だけで構いません。続いて、もしその読みに対して漢字が複数ある場合には、どれを削除するか選びます。

複数の漢字をまとめて削除する場合

まず削除する漢字のデータを用意します。データの形式は、まとめて登録する場合と同じです(4.3.5.1 節「交ぜ書き変換辞書への登録」p.16 参照)。次に、そのデータがあるバッファで `M-x tcode-mazegaki-delete-entries-buffer` とします。なお、辞書に登録されていない読みと漢字との組み合わせに関しては無視します。

入力法を覚えた漢字一字について不要な漢字をまとめて削除する場合

`M-x tcode-mazegaki-delete-kanji-from-dictionary` とし、漢字を 1 字指定します。こうすると、その漢字が読みに含まれず、漢字に含まれるものを削除します。削除する前に、確認のため、漢字一覧が表示されますが、このうち、実際に削除されるのは、指定した漢字を含む候補だけです。なお、このときに表示される漢字一覧はコマンド `tcode-mazegaki-make-entries-region` で登録可能な形式です。部分的に登録し直したい場合に使うとよいでしょう。

ここで、次のことに注意してください。たとえば、「花」を指定した場合、読み「あじさい」・漢字「紫陽花」というように、特殊な読みを持つものも削除されます。このような特殊な読みを辞書に残しておきたいのなら、新たに登録し直さなければなりません。

4.3.6 tc2 での交ぜ書き変換—LEIM を使用する場合

tcode-kkc-region Function

リージョンで囲んだ平仮名を漢字に変換する。

`M-x tcode-kkc-region` とする事で、リージョンで囲んだ平仮名を漢字に変換します。この時リージョン内は文章ではなく、単語を含めるようにして下さい。通常の交ぜ書き変換辞書に載っていないものを変換するのが主な目的です。変数 `tcode-kkc-toroku` で、変換した漢字を交ぜ書き変換辞書に追加するか制御できます。

適当なキーにバインドしておくと便利です。例えば `C-SPC` で変換をしたい場合は、次のコードを '`.tc`' に書いておきます。

```
(define-key ctl-x-map " " 'tcode-kkc-region)
```

Emacs 20 以降では、Mule 機能が本家 Emacs に統合されました。同時に英語以外の言語の入力方法として『LEIM』と呼ばれるパッケージが提供されるようになりました。LEIM は日本語用の変換方法に、`kkc-region`(kkc: Kana Kanji Converter) を採用しています。

`tcode-kkc-region` は LEIM 付属の日本語辞書を使って変換を行います。LEIM 付属辞書の実体は、SKK ver. 8.1 付属の '`SKK-JISYO.L`' ですが、変換エンジンは SKK ではありません。Emacs 21 では、混乱を排す為、ファイル名が '`ja-dic.el`' に変わっています。

4.3.7 交ぜ書き変換辞書の作成

この章では、新たに交ぜ書き変換辞書を作成するための手順を簡単に説明します。

ただし、通常は、これから説明する方法は使う必要がありません。交ぜ書き変換辞書は、初期設定を行えば自動的に作成されるからです。自動的に作成される辞書は、「T コードで入力できる漢字がない」という前提のもとで作成されたものです。この辞書をもとにして、登録や削除を行ってください。

4.3.7.1 交ぜ書き辞書の作成に必要なファイル

交ぜ書き変換辞書を作るために必要なファイルは、パッケージインストール時に、様々なファイルと共に一つのディレクトリに置かれます。そのディレクトリは、tcode-site-data-directory の値を調べれば分かります。

以下のファイルが交ぜ書き変換辞書の作成に用いるファイルです。

'pd_kihon.yom'
: 元となる辞書ファイル。

't225.dat'
't300.dat'
't400.dat'
't450.dat'
't575.dat'
't675.dat'
't900.dat'
't1200.dat'
: t の後ろの数字は、漢字を大体何文字位覚えたかを示しています。

4.3.7.2 交ぜ書き変換辞書の作り方

1. 「完全に打ち方を覚えた漢字の表」('certain') と、「まだ完全には覚えていない漢字の表」('uncertain') の二つのファイルを作ります。

   これは、't*.dat' ファイルを適当につなぎ合わせて作ればよいでしょう。't*.dat' の内容をざっと見て、「大体この中にある漢字は全部覚えたな」と思ったファイルを cat などでつなぎ合わせて、'certain' ファイルを作ります。同様に、「今練習中だけど、まだ完全には覚えていないな」という漢字の含まれるファイル群をつなぎ合わせて、'uncertain' ファイルを作ります。

   > ■ただし、't*.dat' は T コード用のデータですので、TUT コードを利用する場合では、'certain'、'uncertain' を、't*.dat' を使わず自分で用意しなければなりません。書き方は 't*.dat' を参考にしてください。□

   たとえば、't225.dat' と 't300.dat' 中の漢字は全部覚えたが、't400.dat' と 't450.dat' 中の漢字はまだ不完全、それ以外の漢字は全然打てない、という場合には、次のように 'certain' と 'uncertain' ファイルを作ることになります。'certain' と 'uncertain' は、tcode-data-directory で指定されたディレクトリに置きます。(例では~/*tcode*/になっています。)

   ```
   % cd tcode-site-data-directory で指定されたディレクトリ
   % cat t225.dat t300.dat > ~/tcode/certain
   % cat t400.dat t450.dat > ~/tcode/uncertain
   ```

   Perl5 を持っている場合には、'mkcertain.pl' というプログラムを使って、この作業を対話的に行うことができます。

■生成される辞書が、‘certain’、‘uncertain’ で指定した文字および指定しなかった文字によってどう違ってくるのかを簡単に補足します。交ぜ書き変換辞書では、ある漢字 (変換候補) に対して、漢字を含む読みがあり得ます。‘certain’、‘uncertain’ の指定の仕方によって、読み中に現れる漢字が変わります。次の対応になっています。

	読み中にその字が現れる	読み中にその字の読みが現れる
certain	○	×
uncertain	○	○
どちらでもない	×	○

たとえば、「茶飯事」という漢字について、「茶」を ‘uncertain’ に含め、「事」を ‘certain’ に含め、「飯」はどちらにも含めないとすると、読みは「さはん事」、「茶はん事」になります。□

2. M-x tcode-make-mazegaki-dictionary を実行します。

4.4 インクリメンタルサーチ

T コードで文字の入力ができるようになると、インクリメンタルサーチのときにも文字を T コードで入力したくなるのは当然です。tc2 では、isearch を再定義して、インクリメンタルサーチ時に T コードで文字を入力できるようにしています。

C-s (isearch-forward) と打つと、元のバッファが T コードモードだった時にはインクリメンタルサーチのプロンプトに ‘I-Search [TC]’ と表示されます。このときキー入力をすると、それを T コードストロークと解釈してサーチ文字列に加えます。

また、インクリメンタルサーチ機能を日本語用に拡張しています。具体的には、以下の点です。

- 行の折り返しも含めた検索

  文中で、語の途中で改行している場合でも、検索することができます。

- C-w の拡張

  もともとのインクリメンタルサーチでは、C-w で、現在のカーソル位置から 1 語を検索文字列として取り出すという機能がありますが、日本語に対しては、1 文字を対象にします。この主な理由は、日本語では単語の区切りを構文的に判断しにくいためです。けれども、C-w を繰り返し用いることで、入力の手間を削減することができます。

  また、英単語の場合は、そのまま単語が検索文字列に追加され、さらに、語と語の間に改行が入っていても検索できます (ハイフネーションには対応していません)。

ただし、上述の拡張は、インクリメンタルサーチ中で T コードモードになっている場合でのみ有効です。そうでない場合は、期待する動作を行わない場合があります。

インクリメンタルサーチ時の文字入力でも、部首合成変換や交ぜ書き変換が使用できます。ただし、交ぜ書き変換の場合は、一旦文字入力モードに入りますので、変換後、RET を入力してください。

サーチを開始してから T コードモードを切り替えるには、isearch-toggle-tcode コマンドを実行します。デフォールトでは C-\ になっています。このとき、元のバッファの T コードモードも連動して切り替わります。モード切り替えキーはカスタマイズすることができます (5.6.1 節「T コードモード切り替えキーの変更」p.30 参照)。

C プログラムを書いていて、コメントに日本語を T コードで入力したい場合があります。ところが、コメントを記述している途中で関数名や変数名などを探そうとすると、モードの切り替えを行わなければなりません。このように、元バッファの T コードモードに関係なく非 T コードサーチをしたい場合があります。ユーザ変数 tcode-isearch-start-state を 0 に設定するとサーチは常に非 T コードサーチから開始されます (5.4 節「ユーザ変数」p.27 参照)。

4.5 補完機能

tc2 には、入力の補完機能があります。補完機能とは、入力に応じて、その入力した文字列で始まる単語の残りの部分を補う機能です。実際には、補完できる候補がある場合は自動的に表示されますので、その候補でよければそれを選択することになります。

補完機能はデフォールトでは使わないようになっています。使えるようにするには、'.tc' に次のコードを追加します。

```
(add-hook 'tcode-ready-hook
          (function
           (lambda ()
             (require 'tc-complete))))
```

補完機能では、補完用の辞書を用います。この辞書に語を登録しておくことで、先頭の何文字かを入力すると、登録してある語が補完できるようになります。辞書は tcode-data-directory に 'complete.dic' という名前で作成します。

辞書の形式は次のとおりです。1 行に一つの候補を記述します。また、ファイルの先頭に近い方が優先度が高くなります。記述の仕方は 2 通りあります。一つ目は、候補をそのまま記述する方法です。二つ目は、補完の際にマッチするときに使う文字列と補完した後の文字列の両方を空白で区切って記述する方法です。記述例を次に示します。

```
シミュレーション
シミュレータ
四じょうなわて 四條畷
```

上の例で、1 行目と 2 行目は候補をそのまま記述しています。1 行目と 2 行目は、「シミュレー」まで同じ文字列ですが、この場合、「シミュレーション」の方がファイルの先頭に近いので、優先度が高くなります。また、3 行目では、「四じょ……」と入力して補完すると「四條畷」が挿入されます。補完というよりは変換かもしれません。

交ぜ書き変換用辞書も補完に利用できます。ただし、交ぜ書き変換用辞書は補完候補を探すのには向いていない形式ですので、tcode-complete-mazegaki-prefix-length で指定された文字数 (デフォールトでは 3) の文字列に対してのみ候補を表示します。一度補完すると自動的に補完用辞書にその候補を登録しますので、次回以降は、その候補については入力文字数の制約はなくなります。

さて、準備が整ったところで、補完機能を使って実際に補完する方法を説明します。例えば、上に挙げた辞書を使うとします。「シミュ」と入力すると、その続きに、'>シミュレーション [2) シミュレータ]<' と表示されます。これは、第 1 候補として「シミュレーション」があり、第 2 候補として「シミュレータ」があることを示しています。「シミュレーション」を選択する場合は M-RET を入力します。「シミュレータ」を選択する場合は M-2 M-RET を入力します。どちらも選択しない場合はそのまま入力し続けてください。なお、補完候補の表示は、しばらく経つと消えるようになっています。すぐに消したい場合は C-g を入力するとよいでしょう。

変数 tcode-complete-max-candidate-count で補完の際の最大候補数を指定できます。また、変数 tcode-complete-context-length-min と tcode-complete-context-length-max により、入力した文字列のどの範囲を補完のために用いるのか指定できます。辞書の名前を変えたい場合は、変数 tcode-complete-dictionary-name で指定します。

辞書への登録は、リージョンで登録したい語を指定して、コマンド tcode-complete-add-to-dictionary を実行することでも行えます。

4.6 様々なヘルプ機能

tc2 では、様々なヘルプ機能を提供しています。

仮想鍵盤表示機能

まず、『仮想鍵盤』とは何か説明します。T コードでは、1 ストローク目をどう入力するかによって、2 ストローク目を入力したときに挿入される文字が違います。つまり、1 ストローク目の入力に応じて、キーボードの各キーの役割が変化するとみなせるわけです。(ある状態での) 仮想鍵盤とは、その状態で持つ各キーの役割がそれぞれのキーに直接結びついているとみなしたときのキーボード (鍵盤) のことです。

1 ストローク目を入力し、しばらく待つと、ウインドウが開き、次のストロークでどの文字が入力されるか (仮想鍵盤) が表示されます。たとえば、j を押すと次のように表示されます。

```
 革   援   徒   舞     節        -    曹    -    -    -
[員   よ   か   っ   ] く      題 [  制   運   び   公]
[と   し   、   *    ] は      設 [  鉄   現   成   映]
[ド   カ   り   」   ] 田      協 [  多   混   選   以]
```

続いて、g を押せば 'は' が挿入されます。ここで、'-' は無効な入力、'*' は特殊な機能 (上の場合は部首合成変換) を表します。

1 打目を入力してから仮想鍵盤が表示されるまでの時間は、変数 tcode-display-help-delay で指定できます (5.4 節「ユーザ変数」p.27 参照)。

ストローク表表示機能

ストローク表 (1.3 節「ストローク表の見方」p.2 参照) を表示します。4.1 節「使用法」p.7 を参照してください。

文字ヘルプ機能

44 や 55 により、文字の打ち方 (『ヘルプ表』) を表示します (4.1 節「使用法」p.7 参照)。たとえば、「字」という字の打ち方は次のように表示されます。

```
                    字 = ウ, 子
・・・・   ・・・・
・・・・   ・・・・   ●…第 1 打鍵
●・・・   ・・・○   ○…第 2 打鍵
```

ここで、‘字 = {ウ, 子}’ は、その文字を構成する部首を表しています (4.2 節「部首合成変換」p.8 参照)。

ヘルプ表は 99 で消すことができます。また、設定を行えば、自動的に消すこともできます (5.4 節「ユーザ変数」p.27 参照)。

ヘルプ表の表示の仕方を変更することもできます。例えば、音声による読み上げなどを行いたい場合など、視覚的な情報よりも、入力キーストロークの情報を利用したい場合があります。この場合、変数 tcode-help-with-real-keys を t にすることで可能になります。更に、出力の形式は、設定により変更できます。詳しくは関数 tcode-stroke-to-string の説明を見てください。

自動ヘルプ機能

部首合成変換や交ぜ書き変換により文字 (列) を入力したとき、その中に直接入力できる文字があれば、その字のヘルプ表を自動的に表示します。

4.7 その他の補助機能

これまでに説明した機能のほかにも、いくつかの便利な機能があります。ただし、設定をしないと使用できないものもあります。

4.7.1 カーソルの色でモードを表す機能

T コードモードと通常 (非 T コード) モードとでカーソルの色を変えることができます (設定が必要です)。変数 tcode-mode-off-cursor-color および tcode-mode-on-cursor-color により、各モードを表すカーソルの色を指定できます。

この機能は X Window System 上の Emacs でのみ使用できます (他のウィンドウシステムでも使用できるかもしれません)。

この機能を使うには、以下を ‘.tc’ に加えて下さい。

```
(add-hook 'tcode-ready-hook
          (function (lambda ()
                      (and window-system
                           (tcode-enable-cursor-to-change-color)))))
```

4.7.2 コントロールキーを伴わないモード切り替え

モードの切り替えは通常 C-\により行いますが、これを特定の (コントロールキーを使わない) キーシーケンスによって行うことができます (設定が必要です)。

具体的には、SPC・TAB・, のキーを使用し、次のように切り替えます。

非 T コードモードから T コードモードへ

, カーソルの直前の文字により動作が異なります。行頭または空白ならモード切り替え、それ以外ならそのまま , を挿入します。

SPC , スペースを挿入後、モードを切り替えます。

SPC TAB　スペースは、一旦挿入されますが、モードを切り替えた後取り除かれます。

T コードモードから非 T コードモードへ

SPC　直前に行われた動作 (キー入力) により動作が異なります。T コードによる入力の直後 (前置型の交ぜ書き変換中のときを除く) では、モード切り替えを行い、かつスペースを挿入します。そうでない場合は、そのままスペースを挿入します。

SPC TAB　スペースは、一旦挿入されますが、モードを切り替えた後取り除かれます。

少し複雑に見えますが、実際は簡単です。たとえば、T コードモードの時に、'たとえば、Emacs では' と入力しようとしたとします。この場合は、'たとえば、' の後 SPC を入力し、続けて 'Emacs ' を入力します。次に、, の後 'では' と入力すればよいのです。ここで、句読点の直後では、SPC による切り替えでも空白は挿入されません。

SPC や SPC , による切り替えの際に挿入される空白を自動的に削除したい場合は tcode-electric-space-without-inserting を t にしてください。

この機能を使用するには、以下を '.tc' に追加してください。

```
;;; コントロールキーを伴わないモード切り替え
(global-set-key " " 'tcode-electric-space)
(global-set-key "," 'tcode-electric-comma)

(add-hook 'tcode-mazegaki-init-hook
          (function (lambda ()
                      (tcode-set-key " " 'tcode-electric-space))))
```

4.7.3 もう一つの外字入力

T コードで直接入力できない文字を手軽に入力したい時があります。この場合に、コマンド tcode-insert-ya-outset を用いて、T コードで用いるキー以外のキーを利用して、2 ストローク (またはそれ以上) で自分の定義した字を入力することができます (設定が必要です)。

たとえば、T コードモードで、T コードで用いるキー以外のキー [にこの機能を割り当てたとします。[を入力し、次に T コードに用いるキーを入力すれば、変数 tcode-ya-outset-map-list で定義された表に基づき字が入力できます。[をさらに入力すると、tcode-ya-outset-map-list のリストを順次切り替えます。なお、[を入力して 1 秒ほど待つと、仮想鍵盤が次のように表示されるようになっていますので、変数 tcode-ya-outset-map-list での字の並び方を覚える必要はありません。

```
 ¥   †   ‡   ¶       ▼        《   》   【   】   “
[★   ◆   ■   ●   ] ▲        〈 [ 〉   〔   〕   〃]
[☆   ◇   □   ○   ] △        ← [ ↓   ↑   →   §]
[※   ‥   …   ◎   ] ▽        ♪ [ 〒   ♂   ♀   -]      (1/3)
```

設定は、コマンド tcode-insert-ya-outset を適当なキーに割り当ててください (5.6.3 節「T コードモードでのキー割り当ての変更」p.31 参照)。

4.7.4 漢字に対する交ぜ書き変換辞書中の読みの表示

オンラインドキュメントで読み方のわからない漢字があった場合、交ぜ書き変換辞書を利用して、その読みを調べることができます。その漢字をリージョンで指定して、M-x tcode-mazegaki-show-yomi-region とすると、その漢字の読みを交ぜ書き変換辞書から探してきて、もし見つかれば、それを表示します。活用する語に対しては、語幹のみをリージョンで指定し、C-u M-x tcode-mazegaki-show-yomi-region とすることで、その語幹を持つ読みを探すことができます。

4.7.5 ひらがなからカタカナへの変換

入力済みのひらがなをカタカナへと変換できます。M-x tcode-katakana-previous-char でカーソル直前の 1 文字をカタカナへ変換します。また、数引数を与えることによって、point の n 文字前までにあるひらがなすべてをカタカナへ変換します。

1 文字だけ変換した場合、その字は自動ヘルプ (4.6 節「様々なヘルプ機能」p.21 参照) の対象となります。

適当なキーに割り当てて使用するとよいでしょう (5.6.3 節「T コードモードでのキー割り当ての変更」p.31 参照)。

また、これとは別の方法として、コマンド tcode-katakana-preceding-chars を用いる方法もあります。これは、たとえば、この機能を%に割り当てているとすると、%を 1 回入力すると、直前の 1 字がひらがななら カタカナになります。さらに%を入力すれば、2 字前の字がその対象となります。このように、対象文字列を対話的に伸ばすことができます。また、DEL を入力することで、対象文字列を縮めることができます。

4.7.6 区点コード・JIS コードによる文字の入力

入力法のわからない漢字の区点コードまたは JIS コードが分かれば、そのコードにより文字入力ができます。

区点コードで入力するには、M-x tcode-insert-kanji-by-kuten-code の後にそのコードを入力してください。JIS コードの場合は M-x tcode-insert-kanji-by-jis-code です。

これらの方法で入力した字は自動ヘルプ (4.6 節「様々なヘルプ機能」p.21 参照) の対象となります。

4.7.7 句読点自動切り替え

書く文章の内容によって句読点を変えることがある場合、その内容に応じて自動的に句読点を切り替えるようにしておけば、うっかり句読点を混在させてしまうことがふせげます。設定をすれば、この機能が使えるようになります。

設定は以下のように行います。自動切り替えを行う主モードを変数 tcode-auto-identify-kutouten-mode-list で指定します。たとえば、text-mode と tex-mode で自動切り替えを有効にするには、以下のように‘.tc’に記述します。

```
(add-hook 'tcode-mode-hook 'tcode-auto-switch-kutouten)
(setq tcode-auto-identify-kutouten-mode-list
      '(text-mode tex-mode))
```

設定したモードにかかわらず、強制的に自動切り替えを行うには、C-u M-x tcode-auto-switch-kutouten とします。

なお、句読点の判定は変数 tcode-kutouten-regexp-alist に基づき行われます。

4.7.8 zap-to-char の拡張

Emacs では M-z で zap-to-char というコマンドが実行できますが、コマンド tcode-zap-to-char では、これを T コードでも入力できるように拡張しています。

この機能を使用したい場合は、次のとおり '.tc' に記述してください。

```
(global-set-key "\M-z" 'tcode-zap-to-char)
```

4.7.9 入力文字の統計管理

入力された文字の統計情報を記録することができます。

次のコードを '.tc' に記述すれば自動的に統計情報が集められるようになります。

```
(add-hook 'tcode-after-load-table-hook
  (lambda ()
    (when (eq tcode-input-method 'tcode)
              (setq tcode-input-statistics-file-name "tcode-stat")
      (tcode-initialize-input-statistics))))
```

コマンド tcode-list-input-statistics-display を実行することにより、文字ごとに何文字入力したか、またヘルプ表で何文字確認したかが表示されます。

変数 tcode-input-statistics-file-name に名前を設定しなければ、つまり、nil に設定しておけば、統計情報は保存されません。

5 カスタマイズ

ここでは tc2 でカスタマイズできる主な部分を簡単に説明します。もっと詳しく知りたい方はソースファイルを読んでください。なお、本マニュアルでは取り上げませんが、custom パッケージの機能を用いてカスタマイズすることもできます。

5.1 個人設定用ファイル

T コードに関するカスタマイズは '~/.tc' で行います。初期設定の方法については、第 2 章「個人環境の構築」p.4 を参照してください。

ファイル名 '.tc' は変数 tcode-init-file-name で変更できます。この場合、'.emacs' で、その中の (require 'tc-setup) よりも前に設定しなければ有効になりません。

5.2 コード表のカスタマイズ

T コードのコード表 (キーストロークと入力される文字との対応) は、自由に変更して構いません。例えば自分の名前に含まれる漢字が直接入力できない場合など、デフォールトの T コードで直接入力できるけれど自分ではめったに使わない字と入れ替えるとよいでしょう。

変更の例を次に示します。次のように '.tc' に記述してください。

```
(add-hook 'tcode-after-load-table-hook
   (lambda ()
     (when (eq tcode-input-method 'tcode)
       ;; 坂→阪
       (tcode-set-action-to-table '(34 29) "阪")
       ;; ―→…
       (tcode-set-action-to-table '(27 32) "…"))))
```

上の例では、「坂」という字を「阪」に、「―」を「…」にそれぞれ変更しています。''(34 29)' の部分はキーストロークを表しています。数字については 5.6.3 節「入力文字と仮想鍵盤との対応」p.31 を参照してください。

5.3 入力フィルタ

キー入力をもとに文字表から引いてきたデータに対して、さらに変更を加えることができます。たとえば、1 バイト文字から 2 バイト文字への変換などは、このフィルタ機能を利用しています。

フィルタの設定は、変数 tcode-input-filter-functions で行います。フィルタを適用する条件と、フィルタの関数名の組を順に記述します。フィルタは、リストの順に、条件が成立した場合に、適用されます。フィルタ関数は、文字を入力して文字を出力しなければなりません。

フィルタの応用例を示します。

○ SKK 風の前置型交ぜ書き変換

シフトさせて入力することによって、前置型の変換モードに切り替えることができます。シフトを有効にして (4.1 節「使用法」p.7 参照)、次のコードを '.tc' に記述します。

```
(setq tcode-input-filter-functions
  '(((or tcode-katakana-mode tcode-shift-state) . japanese-katakana)
    ((and (boundp 'tcode-bushu-prefix-list)
  tcode-bushu-prefix-list)
     . tcode-bushu-prefix-convert)
    (tcode-alnum-2-byte . tcode-1-to-2)
    (tcode-shift-state . tcode-mazegaki-add-prefix)))
```

5.4 ユーザ変数

ここでは、ユーザが変更できる主な変数を説明します。

パス

tcode-data-directory Variable

T コードの各種データファイルを置くディレクトリ。ディレクトリ名の最後が/で終わっていなければなりません。デフォールトでは '~/tcode/'。今後出てくる変数の中で、値が '~/tcode' で始まるものはすべて、この tcode-data-directory の値を変えることで、そのデフォールト値が変わります。ただし、この変数の値は 'tc.el' ロード時にだけ参照されるので、T コード起動後に tcode-data-directory の値を変更しても意味はありません。

tcode-mazegaki-dictionary-name Variable

交ぜ書き変換辞書のファイル名。デフォールトでは 'mazegaki.dic'。

tcode-bushu-dictionary-name Variable

部首合成変換辞書のファイル名。デフォールトでは 'bushu.rev'。

tcode-record-file-name Variable

T コード使用の統計を記録するファイルのパス名。nil なら記録しません。デフォールトでは '~/.tc-record'。

部首合成変換

tcode-bushu-on-demand Variable

部首合成変換辞書の初期化をいつ行うかを指定します。数字が大きい方がより強い要求があったときにしか初期化を行いません。tcode-bushu-on-demand に指定された数以上の要求が来ると、部首合成変換辞書の初期化を行います。

0	'tc.el' ロード時。
1	T コードモードに入ったとき (デフォールト)。
2	部首合成変換を開始したとき。ストロークヘルプを実行したとき。

tcode-use-postfix-bushu-as-default Variable

nil でないときに、jf で後置型部首合成変換を行い、77 で前置型部首変換を行います。nil のときは逆になります (デフォールト)。

交ぜ書き変換

tcode-use-prefix-mazegaki Variable

前置型の交ぜ書き変換 (4.3.3 節「tc2 での交ぜ書き変換—前置型」p.15 参照) を使う時に t。デフォールトでは nil。

tcode-mazegaki-stroke-priority-list Variable

候補選択時の候補の並べ方を指定します。キーアドレスのリストです (5.7 節「個人用設定ファイルの見本」p.33 参照)。

tcode-mazegaki-fixed-priority-count Variable

学習 (候補の入れ替え) する際の、学習の対象外となる候補の数です (4.3.2.1 節「候補の選択法」p.12 参照)。十分大きな値にしておけば、候補の入れ替えをしなくなります。

tcode-mazegaki-yomi-max Variable

交ぜ書き変換の対象とする読みの長さの最大値を指定します。活用語の場合は活用語尾の長さも含みます。

長くすると変換効率が悪くなるので、長くしすぎないようにしてください。

tcode-mazegaki-enable-inflection Variable

nil でないとき、活用語の変換 (4.3.2.2 節「活用する語に対する変換候補の検索法」p.14 参照) をします。そうでないときは、活用語の変換をしません。

ヘルプ

tcode-display-help-delay Variable

T コードモードで、1 打目を入力してから仮想鍵盤 (4.6 節「様々なヘルプ機能」p.21 参照) が表示されるまでの時間 (秒)。

NEmacs では整数しか指定できませんが、Mule/Emacs では小数も指定できます。

tcode-auto-help Variable

nil でないとき自動ヘルプを行います (4.6 節「様々なヘルプ機能」p.21 参照)。

また、この変数の値を'delete-the-char に設定すると、直接入力できる字は削除します。つまり、表示された打ち方を見ながら再入力する必要があります。

tcode-auto-remove-help-count Variable

ヘルプ用のバッファを自動的に消す関数 tcode-auto-remove-help は、この変数で指定した回数だけ呼ばれるとヘルプ用のバッファを消します。たとえば、次の設定では、T コードモードのトグルを行ったときにヘルプ用のバッファが表示されていたら、そのバッファを消します。

```
;;; モード切り替え時にヘルプ用のバッファを消す。
(add-hook 'tcode-ready-hook
          (function
           (lambda ()
             (setq tcode-auto-remove-help-count 1)
             (add-hook 'tcode-toggle-hook
                       'tcode-auto-remove-help))))
```

上の例について、NEmacs 以外の Emacs では、`tcode-toggle-hook` の代わりに `post-command-hook` を使ってもよいでしょう。

tcode-adjust-window-for-help Variable

`nil` でないとき、ヘルプ用のウィンドウの高さを自動的に調整します。
`nil` でも、もともとウィンドウが分割されていなかった場合では、自動的に調整します。

tcode-help-with-real-keys Variable

`nil` でないとき、ヘルプ表を用いずに、キーストロークによりヘルプを表示します。

インクリメンタルサーチ

tcode-isearch-start-state Variable

T コードインクリメンタルサーチ開始時の T コードモードを指定します。

`nil`	バッファの T コードモードに同期 (デフォールト)。
`t`	バッファの T コードモードと独立。開始時はバッファと同じ。
`0`	バッファの T コードモードと独立。常に非 T コードサーチから開始。
`1`	バッファの T コードモードと独立。常に T コードサーチから開始。

バッファローカル変数です。

tcode-isearch-enable-wrapped-search Variable

`nil` でなければ、行を折り返している日本語も検索します。

その他

tcode-load-immediate Variable

`nil` でないとき初期設定時 (‘`tc-setup.el`’ ロード時) に T コードで使用する Lisp ライブラリをすべてロードします。

tcode-verbose-message Variable

`nil` でないとき様々なメッセージを表示します。

tcode-input-command-list Variable

T コードモードのとき通常の T コード入力を行うためのコマンドのリスト。
たとえば、c-mode の ; (`electric-c-semi`) を、T コードモードでは T コード入力に使いたい場合、そのコマンドをこのリストに追加します。

5.5 フック

tcode-ready-hook Variable

T コードに関する初期化を行った直後に呼ばれます。これは Emacs を起動してから初めて `toggle-input-method` を行った時におこります。
T コードモードでのキー割り当ての変更などに使用します。

tcode-bushu-ready-hook Variable

部首合成変換に関する初期化を行った後に呼ばれます。

tcode-mode-hook Variable

各バッファにおいて初めて T コードを使おうとしたときに呼ばれます。

tcode-toggle-hook Variable

T コードモードを切り替えるたびに呼ばれます。

tcode-after-load-table-hook Variable

文字表を読み出すたびに呼ばれます。正確には、読み出した直後に呼ばれます。
コード入力に対する動作を一部変更する場合などに使用します。

tcode-before-read-stroke-hook Variable

T コードモードで、2 ストローク目以降の入力を入力する前に呼ばれます。
入力待ち状態でカーソルの色を変えたい場合などに使用します。

tcode-mazegaki-init-hook Variable

最初に交ぜ書き変換モジュールをロードするときに呼ばれます。
交ぜ書き変換に関する設定を行うときに使用します。

5.6 キー割り当ての変更

5.6.1 T コードモード切り替えキーの変更

T コードモードの切り替えを行うコマンドは `toggle-input-method` です。tc2 では `C-\` に割り当てられています。

これを他のキーに割り当てるには Emacs の `define-key` を使って普通に設定すればよいのです。たとえば `C-j` にこの機能を割り当てたい時には下のように記述します。

```
(define-key global-map "\C-j" 'toggle-input-method)
```

また、`C-\`への割り当てを取り消すには、以下のように記述します。

```
(global-unset-key "\C-\\")
```

インクリメンタルサーチ時の切り替えキーの変更は次のとおりです。

```
(define-key isearch-mode-map "\C-j" 'isearch-toggle-tcode)
(define-key isearch-mode-map "\C-\\" nil)
```

ただし、インクリメンタルサーチ時の切り替えは、NEmacs の場合、次のとおりです。

```
(setq search-tcode-char ?\C-j)
```

5.6.2 入力文字と仮想鍵盤との対応

tc2 でのキー割り当ては、入力文字 (実際のキー) ではなく、仮想鍵盤で用いるキーのアドレスにより行います。デフォールトでは、次の対応になっています。

- 実際のキー配置

```
 1  2  3  4  5   6  7  8  9  0
 q  w  e  r  t   y  u  i  o  p
 a  s  d  f  g   h  j  k  l  ;
 z  x  c  v  b   n  m  ,  .  /
```

- 仮想鍵盤で用いるキーのアドレス

```
 0  1  2  3  4   5  6  7  8  9
10 11 12 13 14  15 16 17 18 19
20 21 22 23 24  25 26 27 28 29
30 31 32 33 34  35 36 37 38 39
```

この対応の変更は、tcode-set-key-layout で行えます。たとえば、Dvorak 配列用に変更したい場合は、以下を '.tc' に追加します。

```
(add-hook 'tcode-ready-hook
  (function (lambda ()
      (tcode-set-key-layout "dvorak"))))
```

このように、よく使用すると思われるキー配列は、あらかじめ tcode-key-layout-list に登録されています。独自のキー配列を用いたい場合は、この変数に新たに登録してください。

5.6.3 T コードモードでのキー割り当ての変更

T コードモードでのキー割り当ての変更は、関数 tcode-set-key で行います。

たとえば、「T コードモードを抜けてからその文字を入力する機能」を定義し、それを\に割り当てるには、以下を '.tc' に記述します。

```
(add-hook 'tcode-ready-hook
          (function
           (lambda ()
             ;; 「\」でモード切り替え後挿入
             (tcode-set-key "\\" 'tcode-inactivate-and-self-insert))))
```

関数 tcode-set-key は次のように定義してあります。

tcode-set-key *key func* &optional *type* Function

key は割り当てを変更するキーです。*func* はコマンドのシンボルまたは nil です。*type* はキーのアドレスを指定する時などに用います。

5.6.4 コード入力に対する動作の変更

T コードのコード入力に対する、挿入される文字列や機能の変更は、関数 tcode-set-action-to-table で行います。

ここでは、例として、通常 T コードで‘「’を入力するコード (,g のストローク) に対し、鍵括弧の対‘「」’を入力する機能を割り当ててみましょう。,g のストロークは’(37 24) になります。以下を‘.tc’に追加します。

```
(defun tcode-insert-matching-kagikakko (&optional arg)
  "‘「」’を挿入し、その間にカーソルを移動する。
整数 ARG を指定すると、挿入前のカーソル位置から ARG 文字を
‘「’の前に入れる。この場合、カーソルは‘」’の手前である。"
  (interactive "P")
  (insert "「")
  (and arg (tcode-forward-char (prefix-numeric-value arg)))
  (save-excursion
    (insert "」")))

(add-hook 'tcode-after-load-table-hook
          (function
           (lambda ()
             (if (eq tcode-input-method 'tcode)
                 ;; 対応する「」を挿入する。
                 (tcode-set-action-to-table
                  '(37 24)
                  'tcode-insert-matching-kagikakko)))))
```

関数 tcode-set-action-to-table は次のように定義してあります。

tcode-set-action-to-table *sequence value* Function

sequence は割り当てを変更するキーストロークです。キーアドレスのリストまたはキーアドレスを指定します。キーアドレスを指定した場合、続いて SPC を入力した時の動作を変更します。

value は以下が指定できます。

コマンド (symbol)
: そのコマンドを実行します。

関数 (symbol, lambda 式)
: その関数を引数なしで呼びます。

変数 (symbol)
: その変数を評価した結果の動作を行います。

表 (vector) 更にその表に従った動作を行います。

リスト (list)
: 更にそのリストに従った動作を行います。

文字列 (string)
: その文字列を挿入します。

文字 (char)
: その文字を挿入します。

nil
: キーストロークを無効にします。

5.7 個人用設定ファイルの見本

‘.tc’ の記述例を以下に挙げます。ただし、何を設定しているかを理解せずに以下の例をそのまま自分の ‘.tc’ にしてはいけません。初めて T コードを使う場合は、最初は変数 tcode-data-directory を自分の環境に合わせて設定するだけにしておき、徐々に変更していく方がよいでしょう。

```
;;; .tc --- setup file for T-Code       -*-emacs-lisp-*-

;;; 【注意】このファイルの設定は、あくまで設定例であり、
;;; 誰にでもおすすめできる設定ではありません。
;;; 内容を理解した上で設定してください。

;;; ●変数の設定
;; tcode-data-directory を "~/T-Code/" に設定する。
(setq tcode-data-directory "~/T-Code/")

;; 起動時に、T コードで使うすべてのライブラリをロードする。
(setq tcode-load-immediate t)

;; 部首合成変換で後置型を使用する。
(setq tcode-use-postfix-bushu-as-default t)

;; 部首合成変換辞書を tc.el ロード時に読み込む。
(setq tcode-bushu-on-demand 0)

;; 表示するメッセージを最低限にする。
(setq tcode-verbose-message nil)

;; ヘルプ用のウィンドウの大きさを自動的に調整する。
(setq tcode-adjust-window-for-help t)

;; .tc-record に記録しない。
(setq tcode-record-file-name nil)


;;; ● global キーの設定

;; ‘tcode-electric-space’ および ‘tcode-electric-comma’ を使った
;; モードの切り替えを有効にする
(global-set-key " " 'tcode-electric-space)
(global-set-key "," 'tcode-electric-comma)


;;; ● T コードモードに初めて入ったときに行う設定
```

```
(add-hook 'tcode-ready-hook
  (lambda ()
    ;; シフトでの入力で、大文字ではなく小文字で入力する。
    ;; 次の tcode-set-key-layout で設定が行われる。
    (setq tcode-shift-lowercase t)
    ;; キー配置として Dvorak を用いる。
    (tcode-set-key-layout "dvorak")
    ;; T コードで使用する 40 個のキー以外のキーへのコマンドの割り当て
    (tcode-set-key "\\" 'tcode-inactivate-and-self-insert)
    ;; カーソルの色
    (when window-system
      (tcode-enable-cursor-to-change-color)
      ;; 安全のため
      (add-hook 'tcode-toggle-hook
'tcode-enable-cursor-to-change-color))
    ;; T コードモードに最初に入った時に交ぜ書き変換の辞書を読み込む。
    (save-excursion
      (tcode-mazegaki-switch-to-dictionary))))


;;; ●文字表を読んだ後に行う設定
;;;   ○２ストローク以上のストロークのキー割り当て
(add-hook 'tcode-after-load-table-hook
  (lambda ()
    (when (eq tcode-input-method 'tcode)
      ;; T コード用の設定
      ;; 表を一部変更する。
      ;;   坂→阪
      (tcode-set-action-to-table '(34 29) "阪")
      ;; QWERTY での「ke」に変数 tcode-left-paren を割り当てる。
      (tcode-set-action-to-table '(27 12)
 'tcode-left-paren)
      ;; QWERTY での「id」に変数 tcode-right-paren を割り当てる。
      (tcode-set-action-to-table '(17 22)
 'tcode-right-paren))))


;;; ●交ぜ書き変換に関する設定
(add-hook 'tcode-mazegaki-init-hook
  (lambda ()
    ;; 候補選択に使用するキーを２段だけにする。
    (setq tcode-mazegaki-stroke-priority-list
  ;; キー配置
  ;;  0  1  2  3  4    5  6  7  8  9
  ;; 10 11 12 13 14   15 16 17 18 19
  ;; 20 21 22 23 24   25 26 27 28 29
  ;; 30 31 32 33 34   35 36 37 38 39
  '(27 22 26 23 28 21 25 24 29 20
       17 12 16 13 18 11 15 14 19 10))
    ;; 変換対象の長さの設定
    (setq tcode-mazegaki-yomi-max 8)
```

```
    ;; 学習 (候補の入れ替え) をしないようにする。
    (setq tcode-mazegaki-fixed-priority-count 10000)
    ;; スペースで、変換だけでなくモード切り替えも行う。
    (tcode-set-key " " 'tcode-electric-space)))


;;; ●その他の設定

;; ○句読点等の切り替え
;; 標準よりも組み合わせを増やし、かつ「(」や「)」も切り替えるようにする。
(defvar tcode-left-paren "(" "* 開き括弧")
(make-variable-buffer-local 'tcode-left-paren)
(defvar tcode-right-paren ")" "* 閉じ括弧")
(make-variable-buffer-local 'tcode-right-paren)
(setq tcode-switch-table-list
      '(;; デフォールト
        ((tcode-touten . "、")
         (tcode-kuten . "。")
         (tcode-left-paren . "(")
         (tcode-right-paren . ")"))
        ;; 1 バイト系
        ((tcode-touten . ", ")
         (tcode-kuten . ". ")
         (tcode-left-paren . "(")
         (tcode-right-paren . ")"))
        ;; 2 バイト系
        ((tcode-touten . ", ")
         (tcode-kuten . ". ")
         (tcode-left-paren . "（")
         (tcode-right-paren . "）"))))

;;; ○句読点の自動切り替え
;; 切り替えの規準 (正規表現) の指定
(setq tcode-kutouten-regexp-alist
      (list '("[、。]" . 1)
            (if (tcode-nemacs-p)
                '("\\z[,.]" . 2)
              '("\\cj[,.]" . 2))
            '("[,. ]" . 3)))
;; 切り替える主モードを指定 (text-mode latex-mode)
(setq tcode-auto-identify-kutouten-mode-list
      '(text-mode latex-mode))
;; バッファで最初に T コードモードに入ったときに、
;; 句読点を自動的に切り替える。
(add-hook 'tcode-mode-hook 'tcode-auto-switch-kutouten)

;;; .tc ends here
```

6 TUT コードで入力するには

TUT コードは T コードと同様の漢字直接入力法です。tc2 は、TUT コードの入力環境としても使用することができます。ここでは、TUT コード用の設定法と、T コード入力環境 tc2 の様々な機能をどう TUT コード用に割り当てているかを簡単に説明します。

6.1 TUT コード用の設定

TUT コード用の設定は、以下を '.tc' に記述します。

```
;;; 基本設定 (TUT コード)
(setq tcode-default-input-method "japanese-TUT-Code")

;;; カタカナモードのトグルを「'」に割り当てる。
(add-hook 'tcode-after-load-table-hook
          (function
           (lambda ()
             (tcode-set-key "'" 'tcode-toggle-katakana-mode))))

;;; ストローク表を 3 段表示にする。
(setq tcode-help-draw-top-keys nil)
```

カタカナモードのトグルを行うキーは、お使いのキー配列や好みに合わせて変更してください。

以上の設定は特に TUT コードで入力するときに必要となる設定ですが、これ以外の設定項目も T コードと同様に変更できます。

6.2 各機能の割り当て

T コードで使用される機能 (第 4 章「T コードを使う」p.7 参照) は、TUT コードでも同様に利用できます。

最上段のキーの組み合わせは、T コードと同様の割り当てです。数字を入力したい場合は、数字キーのあとにスペースを入力してください。

fj (交ぜ書き変換) は alj、jf (部首合成変換) は ala にそれぞれ割り当てています。これらの機能を、TUT 記号を変更して使用することもできます。たとえば次のように '.tc' に記述してください。

```
(add-hook 'tcode-after-load-table-hook
          (function
           (lambda ()
             (if (eq tcode-input-method 'tutcode)
                 (progn
                   ;; 「k SPC」で交ぜ書き変換
                   (tcode-set-action-to-table 27 'tcode-mazegaki-begin-conversion)
                   ;; 「l SPC」で部首合成変換
                   (tcode-set-action-to-table 28 'tcode-bushu-begin-conversion))))))
```

T コードモードでは、?や!が、それぞれヘルプ・交ぜ書き変換辞書からの漢字の削除の機能に割り当てられています。?や!を入力したいときは、C-q の後にそのキーを押して入力してください。

7 補助ソフトウェア

7.1 打ち方表作成プログラム hasida-table

直接入力できる文字の打ち方を A4 紙 2 枚分の表にしたものを、橋田 (hasida@etl.go.jp) さんが考案されました。これは TEX で処理するものでしたが、これを PS で出力できるようにし、また、どの文字を表に入れるかを簡単にカスタマイズできるようにしたプログラム (Perl スクリプト) を藤原 (makoto@ki.nu) さんが作成されました。

このプログラム (hasida-0.6.tar.gz) は、WWW で入手できます (1.4 節「T コード情報の入手方法」p.3 参照)。プログラムの詳しい内容は、プログラムに付属の文書を参照してください。

7.2 T コード練習プログラム DOGGG

『DOGGG』は端末画面用の T コード練習プログラムです。現在、Sun OS 4.1.*x* と Solaris 2.*x* 上での動作が確認されています。また、Windows95 / 98 などの DOS プロンプト、または WindowsNT のコマンドプロンプトでも動作します。ソースファイルは 'doggg.lzh' という名前でウェブサイト (1.4 節「T コード情報の入手方法」p.3 参照) から入手できます。また、DOS 用のバイナリーパッケージも入手可能です。

7.3 skkinput3 を利用するには

tc2 は Emacs 上でしか利用できませんが、X で動作する日本語入力プログラム skkinput3 を利用すれば、X 上の他のアプリケーションに対しても、T コードを使用することができます。

skkinput-3.0.5 以降 (skkiput-3.0.4 以前は不可) のバージョンに対して使用できます。skkinpt3 がインストールされていれば、コマンド tcinput を実行することにより、T コード用の設定で skkinput3 を起動します。

個人用の設定は '~/.tc-skk' で行います。'~/.tc-skk' では、少なくとも、tcode-data-directory の設定と、次のコードを記述してください。

```
;;; isearch で T コードを使用しない。(skkinput3 では使えない。)
(setq tcode-use-isearch nil)
```

skkinput3 では、tc2 の機能に対して以下の制約があります。

ヘルプ関係は対応していません。

skkinput3 で複数のバッファを表示する機能が実装されない限りは対応する予定はありません。

交ぜ書き変換辞書に対する登録や削除のコマンドは使用できません。

デフォールトでは、交ぜ書き変換用辞書の保存は行わないようになっています。次のコードを '~/.tc-skk' に記述することで、学習による交ぜ書き変換用辞書の変更があった場合に、モードをオフにした時点で保存されるようになります。

```
(add-hook 'tcode-im-end-conversion-hook 'tcode-save-dictionaries)
```

7.4 kinput2 を利用するには

tc2 は Emacs 上でしか利用できませんが、X で動作する日本語入力プログラム kinput2 (ただし、Wnn 用) をカスタマイズすれば、X 上の他のアプリケーションに対しても、T コードを使用することができます。

このパッケージ中の 'kinput2' ディレクトリに、'ccdef.tcode' と 'tc-ki2.el' の二つのファイルがあります。以下、kinput2 で T コード入力ができるようにするための方法を説明します。

1. Emacs を起動し、一旦 T コードモードにしてから、'tc-ki2.el' をロードします。'tc-ki2.el' をロードするには、M-x load-file RET の後、'tc-ki2.el' を指定します。
2. 'rule.tcode' というバッファが生成されていますので、これを適当なディレクトリに保存します。これは、C-x b rule.tcode の後、C-x C-w で保存すればよいでしょう。また、保存したディレクトリに、付属の 'ccdef.tcode' も移します。
3. 'rule.tcode' を EUC コードに変換します。たとえば nkf という漢字コード変換プログラムを利用して、次のようにすることで変換できます。

```
% nkf -e rule.tcode > foo
% mv foo rule.tcode
```

4. 'ccdef.tcode' を編集して、'rule.tcode' のパスを、先ほど 'rule.tcode' を置いたパスと一致するように変更します。

以上です。

kinput2 を使用するときは、'ccdef.tcode' のあるディレクトリで

```
% kinput2 -ccdef ccdef.tcode
```

のように起動してください。

概念索引

記号

A

B

C

D

E

I

J

K

L

M

S

T

U

W

あ

か

さ

た

な

は

ま

ら

関数と変数索引

目次