

Migemo: 日本語のインクリメンタル検索

高林 哲^{†,††} 小松 弘幸^{†††} 増井 俊之[†]

インクリメンタル検索は情報検索やテキスト編集などの用途に広く用いられている。しかし、日本語の入力にはかな漢字変換という障壁があるため、キーボードから1文字入力することに検索を進めていくスムーズなインクリメンタル検索は従来、行うことができなかった。本論文では、指定された読みで始まる単語をコンパクトな正規表現に動的に展開してインクリメンタル検索を行う手法 Migemo を提案する。我々は Migemo の実装および評価を行い、これまで困難であったスムーズな日本語のインクリメンタル検索が実現できることを示す。最後に各種の応用例を紹介する。

Migemo: Incremental Search Method for Japanese Text

SATORU TAKABAYASHI,^{†,††} HIROYUKI KOMATSU^{†††}
and TOSHIYUKI MASUI[†]

Although incremental search is used in a wide range of tasks including information retrieval and text editing, conventional incremental search method cannot handle Japanese texts effectively because Japanese text input requires an indirect input method called Kana-Kanji conversion. In this paper, we introduce a new incremental search method called Migemo to realize the incremental search for Japanese texts. Migemo performs the incremental search by dynamically expanding the input pattern into a compact regular expression which represents all the possible words that match the input pattern. We show how Migemo realize the Japanese incremental search, which is hard to perform effectively so far, through the concrete implementation and evaluation. Various applications are also presented.

1. はじめに

インクリメンタル検索とは、ユーザが1文字入力するたびに検索を進めていく検索手法である。図1に示すような通常のキーワード検索では、ユーザははじめにキーワード全体を入力する必要があるが、インクリメンタル検索では、ユーザが最初の1文字を入力した瞬間から検索が始まる。インクリメンタル検索は、ユーザの入力に従って動的に検索が進行する Dynamic Query⁴⁾ の一種といえる。

Emacs などのテキストエディタでは、テキスト編集を効率的に行う手段として、インクリメンタル検索の機能を備えている⁵⁾。インクリメンタル検索は、テキスト内から目的の情報を探すという情報検索の用途

だけではなく、目的の位置にカーソルをすばやく移動するというポインティングの用途にも広く用いられている。図2に“scheme”という単語をインクリメンタル検索する過程を示す。

この例では、ユーザは検索を開始してから **s c h** の3打鍵で目的のテキスト位置に到達している。インクリメンタル検索を用いれば、難しい綴りを持つ長い単語でも最初の数文字を打つだけで検索できる。

Raskin³⁾ は、インクリメンタル検索を、検索がすばやく行えるだけでなく1打鍵ごとにユーザにフィードバックが返るという点においても優れていると主張している。Raskinはこの考えに基づき、インクリメンタル検索専用のキーを備えたワープロ専用機を設計している。

このように、インクリメンタル検索はテキスト編集の作業を効率的に行うための重要な機能である。しかし、一般に日本語は直接入力が行えないため、インクリメンタル検索をスムーズに行うことができない。

インクリメンタル検索では1文字入力することに検索を進めていくという点が重要だが、かな漢字変換は、ある程度まとまった単位の入力を必要とするため、イ

[†] 株式会社ソニーコンピュータサイエンス研究所
Sony Computer Science Laboratories, Inc.

^{††} 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science of Nara Institute of Science and Technology

^{†††} 東京工業大学情報理工学研究所数理・計算科学専攻
Graduate School of Information Science, Tokyo Institute of Technology

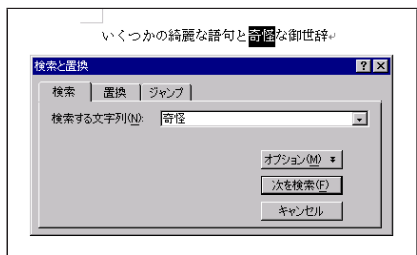


図 1 キーワード検索のダイアログ
Fig. 1 Dialog for keyword search.

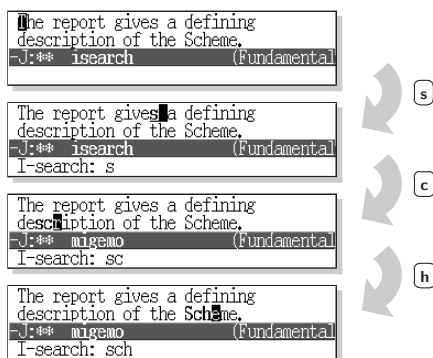


図 2 “scheme” をインクリメンタル検索
Fig. 2 Incremental search for “scheme”.

ンクリメンタル検索との相性が悪い。一般に、かな漢字変換には次の 3 つのステップを要する。

- 読みをローマ字またはかなとして入力する。
- かな表記を漢字表記に変換する。
- 変換候補から適切な単語を選択する。

このため、Emacs ではかな漢字変換を用いて「奇怪」を検索するためには、`[S] [KANA] [k] [i] [k] [a] [i] [CONVERT] [SELECT] [SELECT] [RETURN] [KANA]` のように 10 打鍵以上を要する。図 3 に「奇怪」をかな漢字変換で入力する過程を示す。かな漢字変換をとまなう検索では、単語の読み全体を入力してかな漢字変換を行った後にやっと実際の検索処理が行われるため、インクリメンタル検索の持つ、動的に検索が進行するという利点は完全に失われる。

2. Migemo

前章で述べた問題を解決するために、我々は日本語のインクリメンタル検索の手法 Migemo を考案し、実現した。Migemo は、かな漢字変換のステップを省略

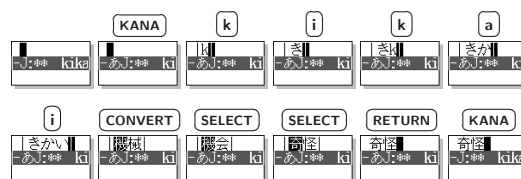


図 3 「奇怪」をかな漢字変換で入力
Fig. 3 Kana-Kanji conversion for 「奇怪」.



図 4 「奇怪」を Migemo でインクリメンタル検索
Fig. 4 Incremental search for 「奇怪」 using Migemo.

し、ローマ字のままの日本語のインクリメンタル検索を実現する。図 4 に Migemo を用いてキーワード「奇怪」をインクリメンタル検索する過程を示す。

この例では、ユーザは検索を開始してから `[k] [i] [k]` という 3 打鍵で目的のテキスト位置に到達している。一方、かな漢字変換をとまなう検索では、前述のとおり 10 打鍵以上を要する。このように、Migemo を用いれば、かな漢字変換の作業に煩わされることなく、スムーズな日本語のインクリメンタル検索が行える。

3. 実 現

Migemo は、ユーザが 1 文字入力するごとに、指定された読みで始まる単語を正規表現に動的に展開してインクリメンタル検索を行う。前述の「奇怪」のインクリメンタル検索に対する正規表現の展開を図 5 に示す。最初の正規表現では“k”にマッチするテキストの位置へ、中央の正規表現では“ki”にマッチする位置へ、最後の正規表現では“kik”にマッチする位置へと、インクリメンタル検索が進行する。

このように、実際の検索は展開された正規表現を用いて行われるが、ユーザからは正規表現を展開する過程は見えないため、通常のインクリメンタル検索とまったく同じように操作が行える。

3.1 正規表現の展開

正規表現の展開は、ヘボン式および訓令式のローマ

[S]: インクリメンタル検索を開始。 [KANA]: かな漢字変換を開始。 [CONVERT]: かな表記を漢字表記に変換。 [SELECT]: 変換候補から適切な単語を選択。 [RETURN]: 変換候補を確定。 [KANA]: かな漢字変換を終了。

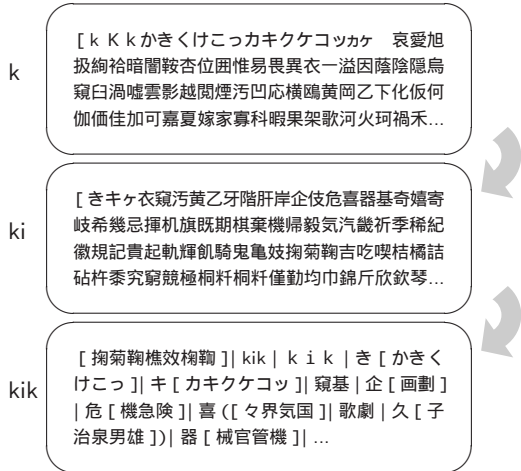


図 5 “k”, “ki”, “kik” に対する正規表現 Fig. 5 Regular expressions for “k”, “ki”, and “kik”.

・きかい	→ 機械 機会 奇怪 器械 貴会 気塊 喜界
・きかいあぶら	→ 機械油
・きかいぶひん	→ 機械部品
・きかいちのう	→ 機械知能
・きかいがっかい	→ 機械学会
・きかいがく	→ 機械学
・きかいがくしゅう	→ 機械学習

図 6 正規表現の生成に用いる辞書 (抜粋) Fig. 6 The dictionary for generating regular expressions (excerpt).

字列をかな表記に変換する規則と、単語辞書を用いて行われる。

かな表記への変換規則による展開については、たとえば図 5 の最初の正規表現では “k” に対して次のような展開が行われている。この展開により、“k” でマッチするすべての平仮名および片仮名をカバーできる。

k ⇒ か | き | く | け | こ | っ | カ | キ | ク | ケ | コ | ツ | カ | ケ

単語辞書を用いた正規表現の展開には、図 6 のような単語辞書が利用される。この辞書を用いることにより、指定された読みで始まるすべての単語にマッチする正規表現を動的に生成できる。Migemo は、入力 “kik” に対して、「きか」、「きき」、「きく」、「きけ」、「きこ」、「きっ」に前方一致でマッチするすべての単語を辞書から取り出して正規表現を生成する。

正規表現を生成する最も単純な方法としては、取り出した単語を or 記号 “|” で連結するというものがある。しかし、マッチする単語を単純に or 記号で連結する手法では、重複の多い巨大な正規表現が生成され

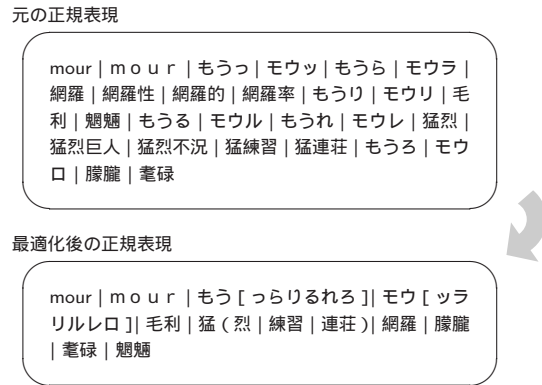


図 7 “mour” に対する正規表現の最適化 Fig. 7 Optimization of the regular expression for “mour”.

るといった問題が起きる。実際、我々の辞書には “k” から始まる単語は約 3 万個が登録されているため、単純に or 記号で結ぶと、186 K バイトもの長さの正規表現が生成される。このようにして生成した正規表現には、

機械油 | 機械部品 | 機械知能 | 機械学会 | 機械学 | 機械学習

のように、候補の各要素の先頭に共通の部分文字列が多く含まれているため、非決定性オートマトンの正規表現エンジンが単純に解釈すると、重複した部分文字列を用いて何度も繰り返してパターンマッチを行うという無駄が生じる¹⁾。これは検索の性能を落とす結果につながる。また、正規表現エンジンの処理系によっては巨大な正規表現を受けつけないものもある。

そこで、Migemo では冗長なパターンを統合して正規表現の最適化を行う。図 7 に、読み “mour” に対する正規表現の最適化を示す。

この例では、元の正規表現に含まれる「網羅」、「網羅性」、「網羅的」といった「網羅」から始まる候補は、長さが最小である「網羅」ひとつにまとめられる。また、「もうっ」、「もうら」、「もうり」なども文字クラスを用いて「もう [っ | ら | り]」とまとめられる。さらに、「猛烈」、「猛練習」、「猛連荘」のように「猛」から始まる単語は括弧でグループ化して「猛 (烈 | 練習 | 連荘)」とまとめられる。最適化のアルゴリズムを以下に示す。

- (1) 先頭の部分文字列 (prefix) が共通する文字列の集合をグループ化する
例: “網羅 | 網羅性 | 網羅的 | 朦朧 | ...”
⇒ “(網羅 | 網羅性 | 網羅的) | 朦朧 | ...”
- (2) 共通する prefix より後ろの部分文字列をグループ化する。

例: “網羅 | 網羅性 | 網羅的” ⇒ “網羅 (| 性 | 的)”

- (a) グループ内に長さゼロの文字列があればグループ全体を捨てる .

例: “網羅 (| 性 | 的)” ⇒ “網羅”

- (b) グループ内のすべての文字列の長さが 1 の場合は文字クラスでまとめる .

例: “も (あ | い | う | え | お)”

⇒ “も [あいうえお]”

- (c) グループ内の文字列の集合に対して同じアルゴリズムを再帰的に適用する .

例: “も (あい | あう | い)”

⇒ “も (あ [いう] い)”

- (3) prefix が共通する文字列の集合がなくなったら終了 .

このようにして正規表現を最適化することにより, “k” のような短い読みに対しても数千バイト程度の正規表現に収められる . たとえば, 単純に or 記号で連結する手法では, “k” の場合 186,446 バイトの正規表現が生成されるが, 最適化を施すことにより 4,994 バイトに縮めることができる .

一方, 正規表現の最適化は, 非決定性オートマトンの正規表現エンジンを用いて検索を行う際に, 冗長なパターンマッチを減らし, 検索の性能を向上させる効果も持つ . たとえば, Pentium III 1 GHz, メモリ 1G バイトの計算機では, 図 7 の正規表現を用いて GNU Emacs 21.2 で 1M バイトのテキスト全体を検索した場合, 元の正規表現では 2.5 秒を要するのに対し, 最適化後の正規表現では 1.1 秒で検索が完了する .

インクリメンタル検索では, ユーザにストレスを与えない高速な処理が不可欠である . そこで, Migemo では, ユーザの入力に高速に追従するために, “a”, “k”, “s” などの短い文字列に対しては, 辞書から動的に正規表現を生成する代わりに, あらかじめ生成した正規表現を用いて, 実用的な速度を実現している . 我々による Migemo の実装では “k” に対する正規表現の生成に前述の計算機で 3.65 秒を要するが, 検索の際には生成済みの正規表現が用いられるため, ユーザは待たされることなくすばやくインクリメンタル検索が行える . 一方, “kik” のように十分に長い文字列の場合は, 動的に正規表現の生成を行っても 0.1 秒以内で処理できるため, ユーザの入力に高速に追従することが可能である .

生成された正規表現でテキストを検索するのに要する時間は, 検索対象が見つかった場合と見つからなかった場合によって異なる . インクリメンタル検索では, カーソル位置から最も近い位置にある候補で検索

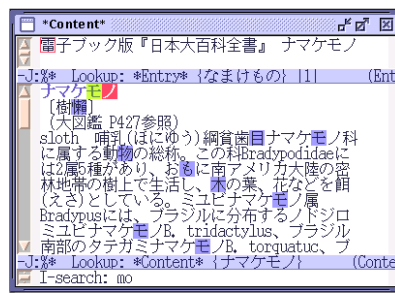


図 8 Emacs 版の Migemo
Fig. 8 Migemo for Emacs.

が停止するため, 検索対象が見つかった場合はカーソルから見つかった位置までの距離に比例する時間で処理が完了する . 一方, 検索対象が見つからない場合にはテキストの終端まで検索する必要があるため, テキストの長さに比例した時間がかかる . 前述のように, 1M バイトのテキストでも 1 秒程度で全体を検索できるため, 通常のテキストであれば十分に高速に検索が行える .

3.2 実装例

我々は, Emacs のインクリメンタル検索機能の拡張として Migemo の実装を行った (図 8) . ユーザは従来のインクリメンタル検索とまったく同じ操作で, 日本語のインクリメンタル検索を行える .

Migemo はフリーソフトウェアとしてインターネット上に公開され, 多くのユーザに利用されている . 現在は我々による Emacs 版の Migemo だけでなく, テキスト Web ブラウザ w3m や テキストエディタ VIM, xzzy といったソフトウェアにも移植されている (図 9) . こうした移植例は Migemo の有効性を裏づけるものといえる .

4. 評価

Migemo の有効性を定量的に評価するために, 日常的に Migemo を利用している 6 人のユーザに 5 日間, インクリメンタル検索の利用状況のログをとってもらい, 分析を行った .

ログにはインクリメンタル検索の利用状況を, 検索に用いられた文字列, 検索して見つかった文字列, 検索の所要時間として記録している . また, インクリメンタル検索の用途を調べるために, Emacs の編集モードの記録も行っている . 編集モードは, C 言語のプログラムを編集しながら c-mode, TeX 文書を編集しながら latex-mode といったように, 編集モードのファイルの種

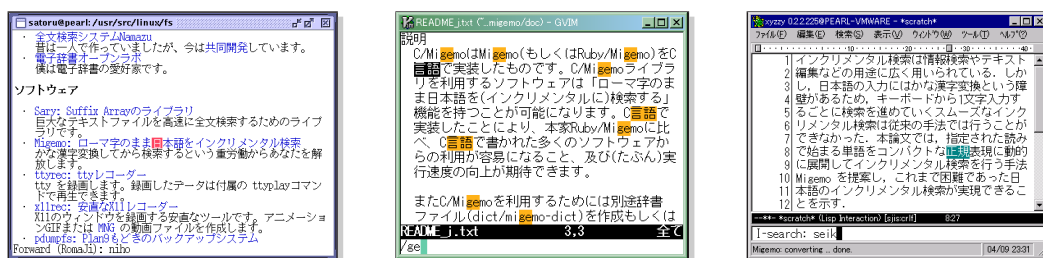


図 9 Migemo の各種実装: w3m, VIM, xzyzy.
Fig. 9 Migemo implementations: w3m, VIM, xzyzy.

表 1 インクリメンタル検索の利用統計

Table 1 Statistics in incremental search use.

ユーザ	英数字インクリメンタル検索			日本語インクリメンタル検索				インクリメンタル検索全体		
	平均回数/日	平均打鍵数	平均時間 (秒)	平均回数	平均打鍵数	平均時間	利用率%	平均回数	平均打鍵数	平均時間
A	126.71	5.32	2.95	55.70	5.14	2.61	30.05	182.41	5.25	2.83
B	72.04	4.50	2.00	3.51	3.92	2.11	5.65	75.55	4.46	2.01
C	105.02	7.79	4.59	4.00	3.75	5.63	3.67	109.02	7.52	4.66
D	406.93	8.66	3.10	23.14	5.10	1.99	5.38	430.07	8.40	3.02
E	113.26	6.22	2.84	50.31	4.75	2.27	30.78	163.58	5.68	2.63
F	111.97	4.39	4.14	4.61	4.50	2.67	3.95	116.58	4.40	4.06

類によって切り替わる。

表 1 に、インクリメンタル検索の利用状況を、英数字、日本語、全体に分けて統計をとった結果を示す。日本語のインクリメンタル検索については全体の利用回数に対する割合として利用率をパーセンテージで求めた。

この表から、ユーザ A は 1 日あたり平均で英数字のインクリメンタル検索を 126.71 回、日本語のインクリメンタル検索を 55.70 回、インクリメンタル検索全体で 182.41 回行っていることが分かる。ユーザ A は、インクリメンタル検索の 1/3 程度が日本語のインクリメンタル検索 (利用率 30.05%) であり、日本語のインクリメンタル検索の利用率が高い。同様にユーザ E も日本語のインクリメンタル検索を利用率 30.78% と多く利用している。これらのユーザは共通して $\text{T}_{\text{E}}\text{X}$ 文書を編集中にインクリメンタル検索を多用している利用形態がログから見て取れた。

一方、ユーザ B, C, D, F は日本語のインクリメンタル検索を利用してはいるものの、その比率は高くない。これらのユーザのログから編集モードを分析すると、ユーザ B は Unix のコマンドライン (shell-mode)、ユーザ C, D は Emacs Lisp プログラミング (emacs-lisp-mode)、ユーザ F は C 言語プログラミング (c-mode)、といった用途で主にインクリメンタル検索を行っていた。Unix のコマンドラインやプログラミングといった場面では日本語が用いられることは少なく、日本語のインクリメンタルの必要性は低い。

これらの結果から、日本語を含む文書を編集する

ユーザは、全体の 1/3 程度のインクリメンタル検索を日本語を対象として行っており、Migemo による日本語インクリメンタル検索が有効に活用されていることが分かる。

5. 議 論

Migemo は、我々が日常的に利用できるレベルの実用性を重視して設計および実装を行った。ここでは Migemo の設計と実装におけるいくつかの特徴を議論する。

5.1 実現手法の方針

Migemo は入力された読みから検索用の正規表現を展開している。一方、日本語のインクリメンタル検索を実現する別の方針として、検索対象のテキストをローマ字に変換するという手法も考えられる。

最も単純な方法としては、検索を行うたびにテキストの内容全体を内部的にローマ字に変換し、そのローマ字の文字列に対して検索を行うというものがある。しかし、この方法はテキストのサイズに比例してローマ字への変換に時間がかかるため、テキストが大きくなると検索の速度の面で破綻する。

ファイルの読み込みのタイミングなどに一括してローマ字に変換する方法や、カーソル位置から検索を進めつつテキストの必要な部分だけを動的にローマ字列へ変換していく方法なども考えられるが、いずれの方法にしても、テキストをローマ字に変換する処理に間違いが起きる可能性が大きい。

たとえば、「博士」という言葉が孤立して現れた場

・い	→	い 位
・いち	→	位置 一
・か	→	か 化
・き	→	き 機
・きかい	→	器械 奇怪 機会 機械
・くし	→	串 櫛
・ない	→	ない 内
・は	→	は 葉
・もう	→	もう 猛
・よう	→	よう 用 要
・れつ	→	列 烈

図 10 本論文のテキストに現れる同音異義語

Fig. 10 Homonyms occurred in the text of this paper.

合, “hakushi” と変換するか “hakase” とするかは人間でも判断できない。一方, Migemo の手法では “hakushi” でも “hakase” でも検索が行える。また, 入力を読みから検索用の正規表現を展開するという我々の手法は比較的容易に実装できるという利点も持つ。

5.2 同音異義語の扱い

Migemo による日本語のインクリメンタル検索では, “kikai” で検索をすると「機械」, 「機会」, 「奇怪」, 「器械」のすべてにマッチするため「奇怪」を検索しているときに「器械」にマッチするという問題が起きる。

しかし, こういった同音異義語が同じテキスト内に近接して現れることは少なく, また不適切なマッチは即座にスキップできるため, 大きな問題にはならない。たとえば, 図表を除いた本論文のテキストの場合, 形態素解析システム茶筌⁸⁾を利用して同音異義語を調べると図 10 しか存在しなかった。このうちの「器械」「奇怪」「機会」「機械」「串」「櫛」「烈」「葉」「猛」「機」などは, 辞書を用いた正規表現の展開を説明する際に現れた単語であり, 本文で自然に使われている言葉には同音異義語はほとんどないことがわかる。

一方, 同音異義語にマッチするという性質には表記の揺れに強いという利点もある。通常のキーワード検索では「こんにちは」で検索すると「今日は」は見つからないが, Migemo で “konnnichiha” で検索すれば両方とも見つかる。同様に, 辞書にさえ登録されれば, “interface” をキーとして, 「インターフェース」, 「インターフェイス」, 「インタフェース」, 「インタフェイス」などの表記の揺れを無視して検索を行うこともできる。

5.3 連文節の検索

現在の実装では単語単位の検索にしか対応していないため, “genzainojissou” で「現在の実装」を検索するような複数の文節にまたがる検索は行えない。

この問題を解決する方法として, 連文節の組合せを考慮して正規表現を展開するというものが考えられる。しかし, 単純にあらゆる組合せを展開すると組合せの数が膨大になるため, 現実的ではない。組合せをおさえるために, 文法に沿った表現だけを生成する方法も考えられるが, その場合でも “karehahakushidesu” から「(彼は(博士|白[紙詩]|薄志)|枯(れ葉|葉)は[串櫛])です」のような複雑な正規表現を生成する必要があり, 文が長くなるにつれて高速な処理が厳しくなる。

そこで, 我々は, より現実的な方法として, インクリメンタル検索しつつ必要に応じてかな漢字変換を行うという手法と, かな漢字変換システム SKK⁶⁾ のようにユーザが文節を指定する手法を考えている。

前者の手法では, 「現在の実装」を検索する際に [g e n z a i n o] まで入力した時点でインクリメンタル検索を一時停止し, 「現在の」をかな漢字変換で確定した後に, [j i s s o u] と入力をしてインクリメンタル検索を再開する。という操作方法が考えられる。後者の手法では, [G e n z a i N o J i s s o u] のようにユーザが文節の先頭を大文字で指定する。この場合は, 文節ごとに正規表現を展開して, それらを連結した正規表現を検索に用いる。

5.4 辞書への依存

Migemo は単語辞書を利用して正規表現を生成するため, 辞書に登録されていない単語は検索できないという性質を持っている。これは, ちょうどかな漢字変換システムが辞書に登録されていない単語の変換を苦手とするのと同様である。

このため, テキストに検索対象が存在しても, 辞書に登録されていない単語だった場合は見つからないという問題が起きる。検索に失敗した場合に, 検索対象が本当に存在しなかったのか, 単語が辞書に登録されていないだけだったのかは, Migemo のシステムとしては区別できないため, ユーザ自身が判断しなければならない。

辞書に登録されていると確信できる一般的な単語の場合は判断に悩むことはないが, 特殊な固有名詞などで判断が難しい場合は, Migemo による検索に失敗した後で, 検索対象が本当に存在しなかったのか通常のキーワード検索で確認する必要がある。辞書に登録されていない単語は検索できないというこの問題は, かな漢字変換の辞書に単語登録するのと同様に, Migemo の辞書に単語登録していけば改善できる。

単語登録をしないで検索する方法としては, 前述の連文節の検索によって, 「契沖(ひさおき)」のような

参 考 文 献

- 1) Friedl, J. E.: *Mastering Regular Expressions*, O'Reilly (1997).
 - 2) Masui, T.: An Efficient Text Input Method for Pen-based Computers, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '98)*, Addison-Wesley, pp. 328–335 (1998).
 - 3) Raskin, J.: *The Humane Interface: New Directions for Designing Interactive Systems*, Addison Wesley (2000).
 - 4) Shneiderman, B.: Dynamic Queries for Visual Information Seeking, *IEEE Software*, Vol. 11, No. 6, pp. 70–77 (1994).
 - 5) Stallman, R.: *GNU Emacs Manual*, Free Software Foundation (2000).
 - 6) 佐藤雅彦: かな漢字変換システム SKK, *bit*, Vol. 23, No. 5, pp. 793–802 (1991).
 - 7) 増井俊之: 検索と例情報を活用した情報管理手法 Q-Pocket, *インタラクティブシステムとソフトウェア VIII: 日本ソフトウェア科学会 WISS2000*, pp. 191–196 (2000).
 - 8) 松本裕治: 形態素解析システム「茶筌」, *情報処理*, Vol. 41, No. 11, pp. 1208–1214 (2000).
-