展開ヘルプ

増　井　俊　之[†]

多くの計算機 OS やアプリケーションにはマニュアルやヘルプ機能が用意されているが、必要な情報をユーザが検索するのが難しいことが多いうえに、適切な機能がみつかった場合でも具体的なパラメタはユーザが指定して操作を行なう必要がある。システムが実行可能な全ての機能をパラメタも含めて自然言語で幅広く表現しておき、それに対応する具体的な実行コマンドも同時に用意しておくことができれば、「**時計␣1:23**」のようなキーワードから「時計を 1:23 にセットする」のような具体的なヘルプ項目を検索すると同時にその実行を指令することが可能になる。

ヘルプ対象システムが提供する全ての機能を正規表現を使って幅広く柔軟に記述し、機能を表現する文字列を順次出力すると同時にユーザが指定したキーワードで効率的な曖昧検索を行なう “Generate and Filter” アルゴリズムにより、現実的な計算量でこのようなシステムを実現することが可能になる。本論文ではこのような「展開ヘルプ」システムの概念と実装について述べる。

ExpandHelp: A Flexible Help Architecture For the Rest of Us

Toshiyuki Masui[†]

Although large amount of manuals and help documents are available on current computing systems, they are not as effectively used as they should be for the following reasons. First, it is difficult for novice users to find the right word for looking up the entry they need. Second, there are mismatches between what the system can do and what the user wants to do. Third, users should translate the abstract description in the help system into concrete user operations.

To make help systems really helpful for users, we developed a flexible help architecture called *ExpandHelp*. ExpandHelp contains a natural language database from which wide range of expressions can be generated so that users can find what they want to do from vague keywords. When a user gives a keyword “1:23” to ExpandHelp, it will display menu items like “Set the system clock to 11:23”, “Set the alarm clock to 1:23”, etc. for the user to find and execute what he wants to do. This is made possible by using the *generate-and-filter* algorithm applied to the help database consisting of a set of regular expressions for generating help texts and corresponding command patterns.

1. はじめに

計算機や家電機器の使い方を調べようとするとき、マニュアルやヘルプシステムから有用な情報が得られることは少なく、ネットで聞いたり検索したりする方が成功することが多い。現状のヘルプシステムには以下のような問題が存在する。

語彙問題　検索に有効な単語や文を選ぶことは難しい[2)]。適切な検索語や綴りがわからない場合もあるし、問題を表現する方法すらわからない場合もある。例えばユーザが「時計を 7 時にセットする」という文字列で検索を行なったとき、ヘルプシステムに「時刻を設定する」というエントリしか存在しなければ検索は成功しない。Mac の画面上部の帯状部分の表示について調べたいとき、この部分が「メニューバー」という名前であり、かつ右側部分は「ステータスバー」と呼ばれることを知らなければ検索ができない。「画面右上の図形表示」のような表現でも検索できるようにする必要があるかもしれない。

ユーザモデル問題　初心者が計算機に期待することと計算機が実行可能な機能はかなり異なっている。初心者は計算機に何ができるかを知らないし、設計者と異なる概念で計算機を見ている可能性が高い。「デスクトップのアイコンを整理する」方法はヘルプ文書に書かれているかもしれないが、ユーザは「パソコン画面を奇麗にする」以外の言葉を思いつかないかもしれない。

用語や概念を理解している場合でも、「2 年以上

† 慶應義塾大学環境情報学部
Faculty of Environment and Information Studies, Keio University

前の大きな動画を捨てる」のように一見簡単に見える作業のやり方がわからないことも多い。
こういった要望のために「逆引き事典」が多数出版されているが、あらゆる要望に答えることはできないし、事典の検索も容易ではない。

パラメタ問題　ユーザが時計を 1:23 にセットしたいとき、ユーザが「時計をセット」のようなヘルプ項目の検索に成功したとしても作業はそれで終わりではなく、時計をあわせる方法を理解した後で実際に時計を 1:23 にあわせる操作を実行しなければならない。通常はユーザの目標は時計を特定の値にセットすることであり、時計をあわせる方法を知ることが目的ではないのだから二度手間である。

Generate and Filter による展開ヘルプ

Web の情報やパソコン内データを検索するために様々なテキスト検索システムが利用されている。計算機や機器の利用に関して疑問な点があれば、とりあえずネットで検索すると必要な情報を得られることが多い。十分巨大で有用なテキストデータベースが存在すれば、既存のテキスト検索手法を利用してそれなりに有効な検索が可能なのは確かである。

もし、あるコンテクストにおいてシステムが実行可能なことすべてを幅広い表現でリストしたテキストデータベースが存在すれば、そのデータに対して通常のテキスト検索手法を適用することにより、ユーザにとって有効なヘルプ情報を得ることができるはずである。例えば、ヘルプデータベースの中に「時計を 6 時 23 分にセットする」「時刻を 7:12 に設定する」のような多様で具体的なテキストが大量に存在すれば、一般的なテキスト検索手法を利用して「**時刻**」「`7:12`」のようなキーワードからヘルプ項目を容易に捜し出すことができ、同時に具体的な時刻設定を実行することが可能になる。また、テキストの曖昧検索を行なうことにより、データの中に「時計」や「時刻」のような文字列しか存在しない場合でも「**時間**」「`7:12`」のようなキーワードから「時刻を 7:12 に設定する」のようなヘルプ項目を選択することが可能になる。

あらゆる表現を含むヘルプデータベースをテキストデータとしてあらかじめ作成しておくことは現実的でないが、幅広い表現を機械的に生成することは難しくない。前述のような問題の場合、「**(時刻|時計|時間) を ([1-9]?[0-9]) 時に (セットする|設定する|あわせる)**」のような正規表現を用意することによって「時計を 6 時にセットする」「時刻を 7 時に設定する」のような多様な表現を生成することができ、生成されたテキストを検索対象とすることができる。このとき、文字列を生成するのと同時にユーザが指定した検索文字列でフィルタリングを行なう “**Generate and Filter**” を行なうことにより、実際にすべてのテキストを展開出力することなく効率的に検索を行なうことができる。

生成されたエントリ項目に対してそれを実行する方法も記述しておけば、検索されたヘルプ項目を選ぶと同時に実行を指令することができる。また、文字列検索手法として曖昧パタンマッチアルゴリズムを利用すれば、検索文字列がヘルプ項目の記述と多少異なっていた場合でも項目をみつけることができる。このような技法を統合したヘルプシステムの手法を「**展開ヘルプ**」と呼ぶことにする。

パズルを解く手法として、すべての解のパタンを生成しながらそれが条件を満たすかどうか判定する “Generate and Test☆” という手法が知られている。たとえば 8-Queen 問題を Generate and Test で解こうとする場合、あらゆる Queen の配置を生成しながらその配置が問題の条件を満たしているかを判定し、条件を満たしているものを解として出力する。8 個の Queen を盤面に配置するには ${}_{64}C_8$(≃44 億) 通りの置き方があるため、これらを素直にすべて生成してから判定を行なうと大量の計算時間が必要であるが、条件にあわない配置は生成の初期段階で無視するような工夫をすることによって高速に解を求めることができる。

2. 展開ヘルプ利用例

Mac のメニューバーに常駐するアプリケーションとして実装した展開ヘルプの利用例を示す。

メニューバーの「ExHelp」というアイコンをクリックすると図 1 のように検索文字列入力枠が表示される。

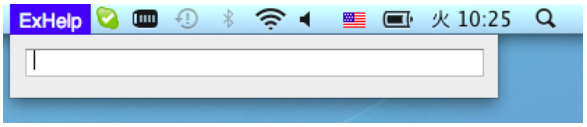

図 1　展開ヘルプの起動.

入力枠に「`3`」を入力すると、システムが実行可能な機能のうち「3」を含むものがリストされる。その時点でのファイル名やプロセス名で「3」を含むものに関する機能、パラメタに「3」が含まれる可能性のある機能、時刻に関連する機能など、膨大なリストの一部が表示されている。

☆ http://en.wikipedia.org/wiki/Generate_and_test

図 2　検索パタン「3」を入力.

ここで「junk3 というファイルを Emacs エディタで開く」を選択すると、図 3 のように Emacs が起動して junk3 というファイルの編集画面になる。

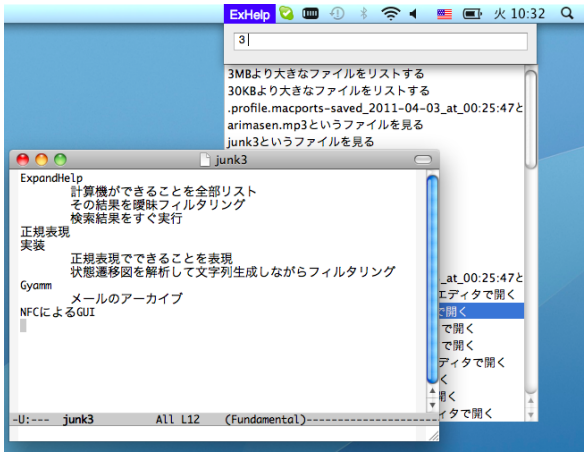

図 3　Emacs を起動して junk3 を編集.

「**鎌倉**」と入力すると、鎌倉に関する各種のヘルプエントリが表示される。エントリを選択するとブラウザが開いて天気予報や時刻表ページが表示される。

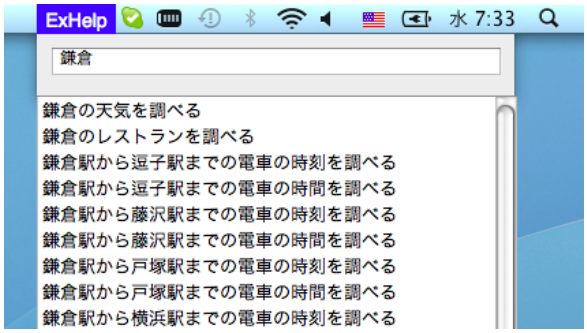

図 4　「鎌倉」で検索

junk.html という名前のファイルを Firefox で開こうとして「`jumk giref`」のように間違ったスペルを入力した場合でも、自動的に曖昧検索が実行されて図 5 のように目的のエントリが表示される。

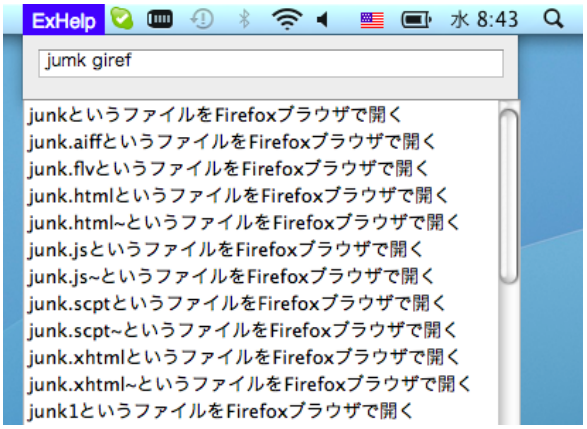

図 5　スペルミスに対する曖昧検索.

「`3:45`」のように具体的な時刻をキーワードとして指定すると、図 6 のようなヘルプ項目が表示され、項目を選択すると実際にシステムの時間が 3:45 にセットされる。別の任意の時刻を指定しても同様である。

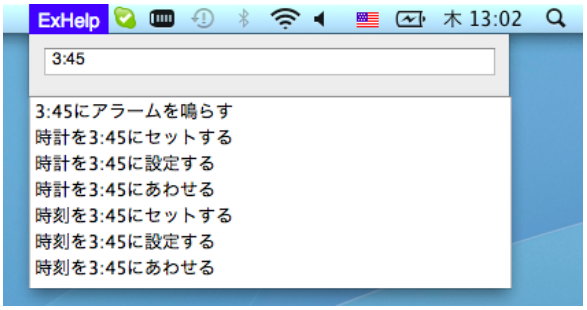

図 6　3:45 に時刻をセット

「**新** `omni`」のようなキーワードを指定すると図 7 のようなヘルプ項目が表示され、項目を選択すると Spotlight の機能を利用して最新の OmniGraffle のファイルが検索されて OmniGraffle でオープンされる。

図 7　最新の OmniGraffle ファイルをオープンする

Mac では NSMetadataQuery ライブラリを利用し

てファイルの全文検索や属性検索を行なうことができるが、属性を指定して検索するための簡単な方法が用意されていないためこの機能は活用されていない☆。ユーザの頭に浮かぶ平易な表現と計算機コマンドの対応をヘルプ項目として用意しておくことにより、図 7 のような直感的にヘルプ検索とコマンド実行を行なうことができる。

このように、ヘルプ項目が存在する限り、コマンド名やパラメタ値やその他の属性を簡潔にキーワードとして指定することによってユーザが実行したい具体的な機能を検索することができ、それを選択してその場で実行させることができることになる。

3. 展開ヘルプの実装

展開ヘルプでは、正規表現で記述された多様な表現のヘルプ項目を Generate and Filter で効率的に検索することができる。これらの実装について以下に述べる。

3.1 ヘルプ項目の記述

ヘルプ項目は検索の対象となる文字列を表現する正規表現とそれに対応したコマンドパタンの組で構成され、ヘルプデータベースはヘルプ項目の集合として表現される。ユーザからの様々なクエリに対応するため、できる限り多様な表現を用意しておく。例えば時刻を設定するヘルプ項目として以下のようなものを用意する。

ヘルプエントリの正規表現
(時計|時刻)を(1?[0-9])時に(設定|セット)する
実行コマンド
setdate {$2}:00

この正規表現は「時計を 0 時に設定する」「時計を 1 時に設定する」... 「時刻を 19 時にセットする」というヘルプ文字列の集合を表現しており、各ヘルプ文字列に「`setdate 0:00`」「`setdate 1:00`」...「`setdate 19:00`」のようなコマンド文字列が対応している。ユーザが「**時間_7**」のようなキーワードを指定すると「**時間**」が「時計」と曖昧マッチし、「7」が 2 番目の括弧「`(1?[0-9])` に」マッチするので「時間を 7 時にセットする」のようなエントリが表示され、それを選択すると「`setdate 7:00`」が実行される。

☆ NSMetadataQuery クラスを利用するとファイル属性やソート順を指定した検索ができるが、ContentType として “`com.omnigroup.omnigraffle.graffle`” を指定したり、ファイル修正時刻をもとにしたソートを指定するといった複雑な指示が必要である。

この例ではパタンに利用される文字列 (e.g. 「7」) をコマンドでも利用しているが、パタンと異なる文字列をコマンド内で利用する必要がある場合は以下のようにタブ文字で区切ってパラメタを指定する。

ヘルプエントリの正規表現
(鎌倉\t773|秋葉原\t682)の天気を調べる
実行コマンド
open http://tenki.jp/forecast/point-{$1}.html

この場合「鎌倉」「天気」のようなキーワードを指定すると「鎌倉の天気を調べる」というヘルプエントリが表示され、それを選択すると「`open http://tenki.jp/forecast/point-773.html`」が実行される。

正規表現は静的なテキストとして記述する必要はなく、実行時のコンテクストやデータベースなどから動的に生成することもできる。

3.2 正規表現の展開と検索

すべてのヘルプ項目の正規表現はひとつにまとめられ、展開ヘルプが扱うあらゆる文字列を表現するひとつの状態遷移機械に変換された後、ユーザに指定されたパタンとのマッチングを行ないながら状態遷移を表現する木構造とテキストを生成していく。

3.2.1 状態遷移機械の生成

正規表現は状態遷移機械と等価であり、機械的に変換することができる。たとえば、「**時 (間|刻) を (1|2|3) 時に**」という正規表現パタンからはそれと等価な図 8 のような状態遷移機械が生成される。

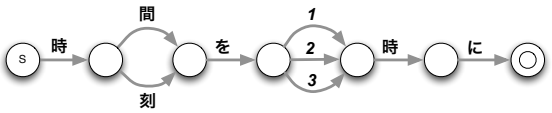

図 8 時刻指定の状態遷移機械.

3.2.2 テキストの生成

正規表現から生成された状態遷移機械のノードを順番にたどることにより、もとの正規表現が表現するあらゆるテキストを生成することができる。

例えば図 8 の場合、開始ノード (ⓢ) からスタートし、エッジをたどって図 9 のような木を幅優先的に作成しながらテキストを生成する。最初の世代では「時」という文字列が生成されて次の世代の計算に移る。エッジをひとつたどった次の世代では「間」「刻」のような文字列が追加され、「時間」「時刻」のような文字列が生成される。これを繰り返すことにより、も

との正規表現に対応するあらゆる文字列を生成することができる。

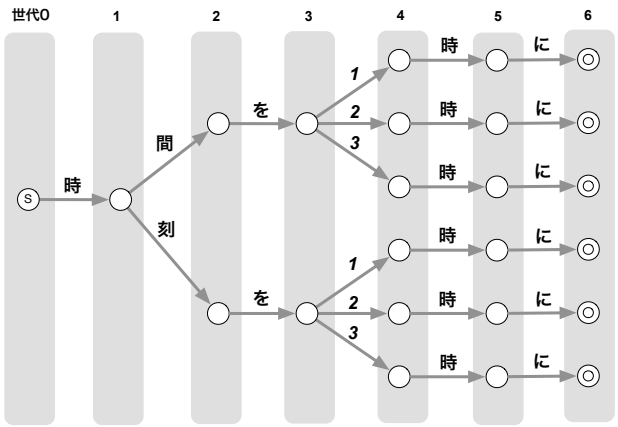

図 9 状態遷移機械をたどって木構造を生成.

「(0|1)+」のように「+」や「*」などを含む正規表現の場合はループを含む状態遷移機械が生成され、エッジをたどりながら木を生成すると無限に深い木が生成されてしまうので、ループの展開回数には制限を設けている。

3.2.3 Generate and Filter

展開ヘルプでは「シフタアルゴリズム[4]」を利用することにより、正規表現からテキストを生成すると同時に曖昧パタンマッチを実行する Generate and Filter を実現している。

シフタアルゴリズム

シフタアルゴリズムは、ビット演算を利用した単純で高速なパタンマッチアルゴリズムであり、最小限のメモリで高速に曖昧検索を実行できるという特徴を持っている。

「restaurant」というパタンを認識する状態遷移機械は図 10 のように表現できる。ここで、灰色のノードはアクティブな状態を示している。

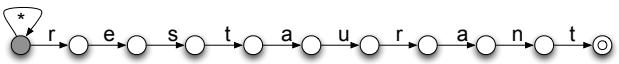

図 10 「restaurant」を認識する状態遷移機械.

最初は左端のノードだけがアクティブになっているが、「r」を認識すると二番目のノードもアクティブになり、図 11 の状態に遷移する。

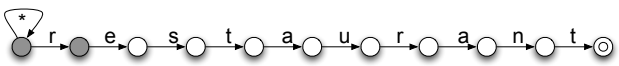

図 11 「r」を認識した後の状態.

「restaurant」を認識すると、状態は図 12 のように変化する。

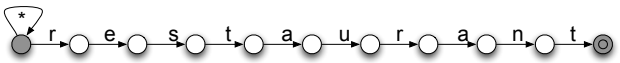

図 12 「restaurant」を認識した後の状態.

シフタアルゴリズムではノードのアクティブ状態をビットパタンで表現する。図 10 の状態は 10000000000 と表現し、「restaurant」認識後の図 12 の状態は 10000000001 と表現する。この状態遷移機械は非決定性であるが、状態の表現に 10 ビットしか必要としないため、状態をひとつの整数値で表現できることに加え、状態遷移を簡単な論理積演算とシフト演算だけで高速に計算できるという特徴がある。

シフタアルゴリズムによる曖昧パタンマッチ

図 10 の状態遷移機械は「restaurant」というパタンを正確に認識するためのものであるが、これを拡張することによって曖昧検索を実行できる状態遷移機械を構築することができる。たとえば曖昧検索によって「ristorante」という文字列が「restaurant」に近いということを認識することができる。

図 13 「restaurant」と「ristorante」の差異.

図 10 の状態遷移機械に状態を追加して図 14 のように拡張することにより、検索単語「restaurant」に対して 0〜3 個のミスマッチを許す曖昧検索を実行することができる。

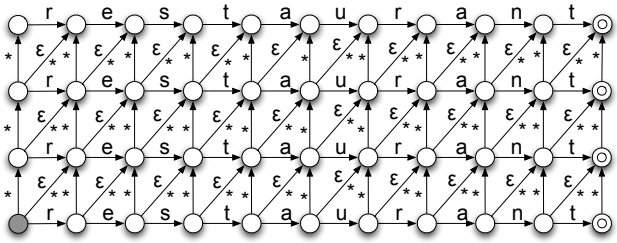

図 14 0〜3 個のミスマッチを許すパタンマッチャ.

最下行の右端の◎は入力が完全に「restaurant」にマッチしたときの遷移先であり、下から二番目の行の右端の◎は 1 文字誤りを含んでマッチしたときの遷移先を示している。「*」(あらゆる文字にマッチ)、「ϵ」(空文字にマッチ) による縦と斜め方向の遷移を追加していることにより曖昧マッチングが可能になっている。

最初は左下のノードだけがアクティブになっているが、「r」以外の文字が入力されたときは「*」と書かれた遷移がアクティブとなり、接続されたノードがアクティブになる。同時に「ϵ」と書かれた遷移も自動的にアクティブになる。これらの組み合わせにより、1 文字余分/1 文字不足/文字置換に対する曖昧検索が可能になっている。

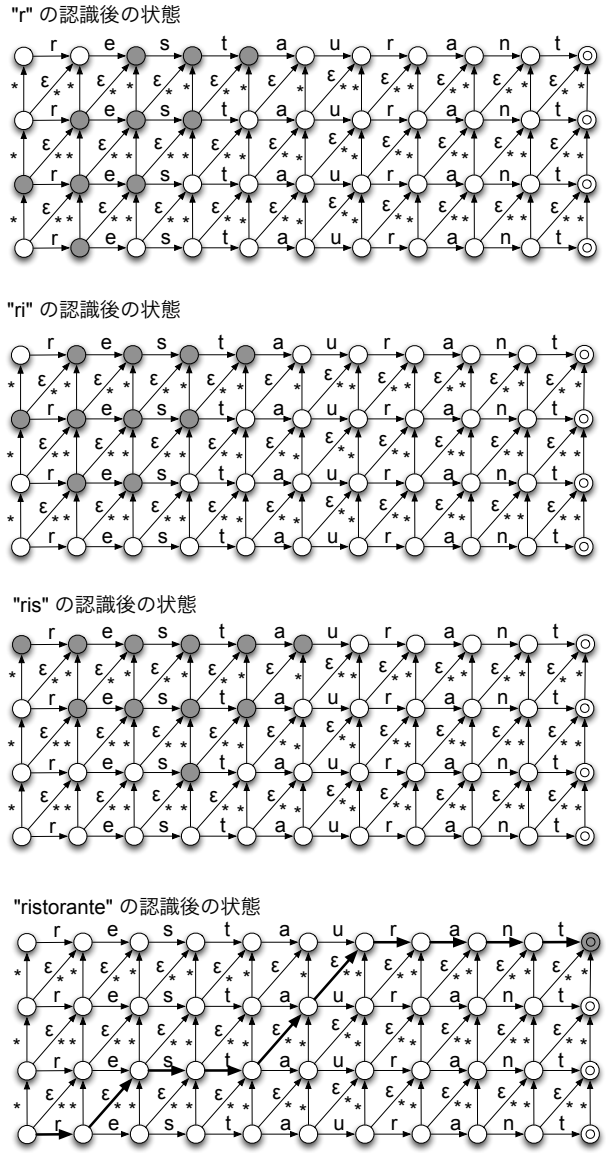

図 15 「restaurant」を認識するマッチャに「ristorante」を入力.

この状態遷移機械に「ristorante」という文字列を入力したときの様子を図 15 に示す。「ristorante」による遷移が終了すると、右上のノードだけがアクティブになっているので、「ristorante」と「restaurant」は 3 文字誤りでマッチすることがわかる。

図 15 の状態遷移は複雑に見えるが、4 個の整数変数の間の論理演算を行なうだけで遷移を計算しているため高速でメモリ効率も良い。

パタンマッチャを Generate and Filter で利用

展開ヘルプでは、図 8 のような状態遷移機械を幅優先的に図 9 のように木構造に展開するのと同時に、ユーザから与えられた検索文字列とそれまでに生成された文字列とのマッチングを行なってシフタアルゴリズムのマッチング状況を木のノードに格納する。

ユーザから「**時計␣3 時**」☆という検索パタンが与えられたときは図 16 のような状態遷移機械を利用する。

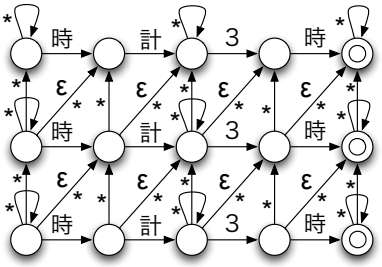

図 16 「**時計␣3 時**」のパタンマッチャ.

この状態遷移機械の初期状態での各ノードのアクティブ状態は図 17 のようになる。シフタアルゴリズムによる表現では {00100, 01000, 10000} となるので、この値を開始ノードⓢ に格納しておく。

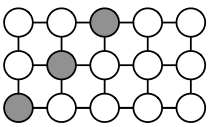

図 17 初期状態のアクティブ状態.

この状態遷移機械が「時」を認識すると、ⓢ に格納された値と入力文字「時」からマッチング状態が計算されてアクティブ状態が図 18 のように変化し、この値 {01110, 11100, 11000} を 2 番目のノードに格納する。

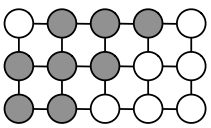

図 18 「時」を認識した後の状態.

このように、各ノードのマッチング状態は遷移前のノードの状態と遷移文字のみから計算されてノードに保存される。状態遷移機械を木構造に展開して文字

☆ 空白文字 (「␣」) は正規表現の「.*」の代用

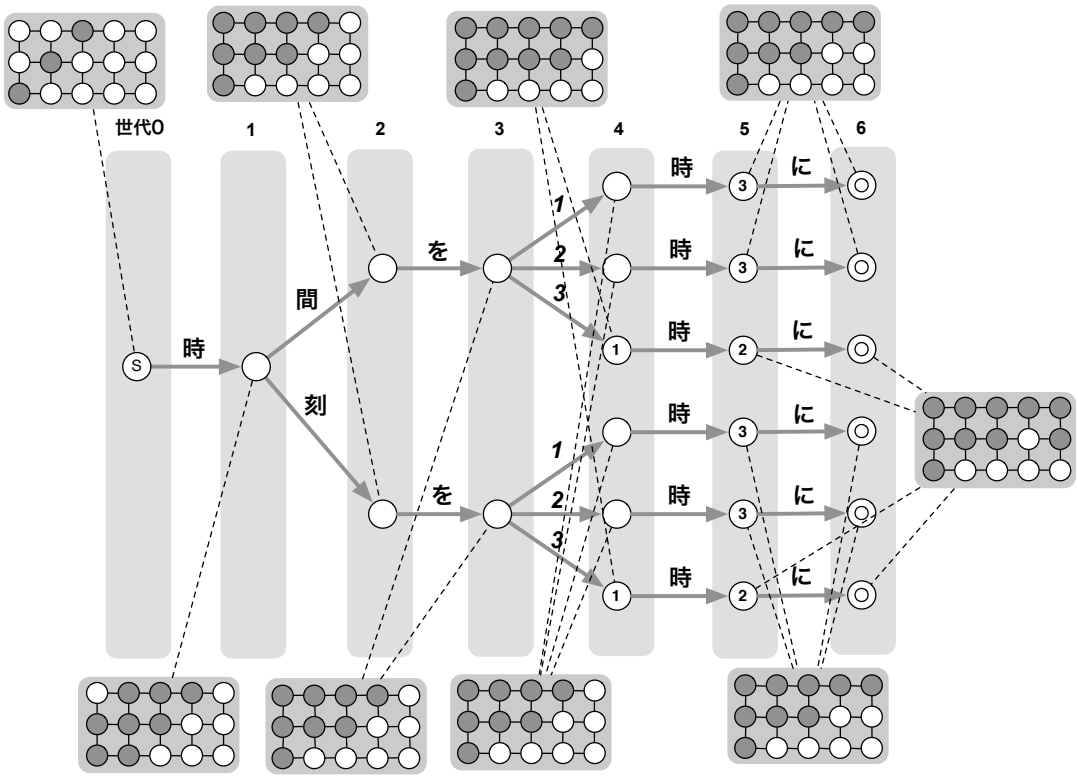

図 19 「**時計_3 時**」へのマッチ状態を計算しながら木を生成.

列を生成するのと同時にシフタアルゴリズムのマッチングを実行し、その結果のマッチングパタンを出力される木構造のノードに格納することにより、文字列の出力とフィルタリングを同時に行なう Generate and Filter が実現されることになる。

図 16 のマッチャに対して「**時計_3 時**」というパタンが与えられたときの Generate and Filter 実行により作成される木構造を図 19 に示す。ノード① のマッチ状態を見るとシフタアルゴリズムのマッチャの右上のノードがアクティブになっているため、「時間を 3」および「時刻を 3」という文字列が「**時計_3 時**」というパタンに曖昧度 2 でマッチしていることがわかる。また② では最右桁の上から二番目のノードもアクティブになっているため、「時間を 3 時」および「時刻を 3 時」という文字列が「**時計_3 時**」に曖昧度 1 でマッチしていることがわかる。

パタンとコマンドの表現

ヘルプ項目の正規表現及びそれに対応するコマンドの MacRuby による記述の例を図 20 に示す。パタンとして`#{ls}`, `#{ps}` などと書かれた部分は Ruby で定義したメソッドで、その時点でのファイルやプロセスを正規表現としてリストする。

4. 議　　論

正規表現記述の利用について

展開ヘルプのエントリを記述する人は正規表現を記述する必要がある。複雑な正規表現の理解は難しいが、様々な表現を「|」で併記する程度であればそれほど難しくないであろう。正規表現の記号を全く利用しなくてもエントリの記述は可能である。

集合知の利用

最初に与えたキーワードでヘルプエントリがみつからず、別の手段で答を発見することができた場合、最初にユーザが与えた文字列を含むヘルプ項目をユーザが自分でデータベースに追加できるようにしておけば後々役立つことがあるだろう。Wiki などを使って集合知をヘルプデータに反映させることができれば有効であろう。

関 連 研 究

インテリジェントなヘルプシステムに関しては数多くの研究が存在するが[1)]、現在広く利用されているものはほぼ存在せず、研究は下火になっているようである。これはインターネット上で検索したりチャットや SNS などで質問したりする方が有効なことが一因であろう。ネット上で手軽にヘルプ情報を検索したり他人に意見を聞けるようになったことは喜ばしいことであるが、何でも他人に聞く人間は嫌われるし、ネットで問題解決方法がわかった場合でもそれを直線実行できないという問題が存在する。

Little のシステム[3)] は、「`left margin 2 inches`」のようなキーワードから「`ActiveDocument.PageSetup.LeftMargin = InchesToPoints(2)`」のようなコマンド文字列を生成することができる。システムは提供する関数や引数に関する完全なデータベースを持っており、

```
['(鎌倉\t773|平塚\t772|藤沢\t774|秋葉原\t682|渋谷\t694|横浜\t744)の天気を調べる',
 '`open http://tenki.jp/forecast/point-#{$1}.html`'],
['(鎌倉\tkanagawa/A1404/A140402/R2568/|湘南台\tkanagawa/A1404/A140405/lst/)のレストランを調べる',
 '`open http://r.tabelog.com/#{$1}`'],
['(時計|時間|時刻)を(0|1|2|3|4|5|6|7|8|9|10|11|12)時に(セットする|設定する|あわせる)',
 '# date #{$2}:00',
 /[0-9]/], # 数字を入力したときだけ利用
['(0|1|2|3|4|5|6|7|8|9|10|11|12)時に(時計|時間|時刻)を(セットする|設定する|あわせる)',
 '# date #{$1}:00',
 /[0-9]/],
['(時計|時間|時刻)を(0|1|2|3|4|5|6|7|8|9|10|11|12):(0|1|2|3|4|5)(0|1|2|3|4|5|6|7|8|9)(AM|PM)に(セットする|設定する|あわせる)',
 'setdate(#{$2},#{$3},#{$4})',
 /[0-9]:[0-5][0-9]/],
['(0|1|2|3|4|5|6|7|8|9|10|11|12):(0|1|2|3|4|5)(0|1|2|3|4|5|6|7|8|9)にアラームを鳴らす',
 'alarm(#{$2},#{$3},#{$4})',
 /[0-9]:[0-5][0-9]/],
['(#{ps})というプロセス|アプリケーション|プログラム)を(消す|止める|停止する|殺す|落とす)',
 '`kill -9 #{$1}`'],
['(走って|動いて)いる(プロセス|アプリケーション|プログラム)をリストする',
 '`ps -eaf`'],
['現在のディレクトリのファイルをリストする',
 '`terminal "ls -l"`'],
['(#{station})駅から(#{station})駅までの電車(の(時刻|時間))を調べる',
 '`open "http://www.jorudan.co.jp/norikae/cgi/nori.cgi?rf=top&eok1=&eok2=&pg=0&eki1=#{$1}&eki2=#{$2}&Cway=0&S=検索&Csg=1"`'],
['(増井\tmasui|池田信夫\tikedanob)のtwitterを読む',
 '`open http://twitter.com/#{$1}`'],
['Firefox(ブラウザ)を(起動する|動かす|使う|走らせる)',
 '`open -a firefox`'],
['(#{ls})というファイルをFirefox(ブラウザ)で開く',
 '`open "#{$1}" -a firefox`'],
['(最新の|(最も|いちばん)新しい)(#{apps})ファイルを(開く|見る)',
 'file=`mdfind2 -c #{$3} -f 1970 | head -1`; `open \\#{file}`'],
['大きな(#{apps})ファイルをリストする',
 '`terminal "mdfind2 -c #{$1} -b 10 | head -10"`'],
```

図 20 MacRuby による展開ヘルプデータの記述例.

その知識とキーワードにもとづいてコマンドを生成する。このようなユーザ補助はプログラマにとって非常に便利であるが、コマンドの文法を文脈自由文法で正確に記述しておく必要があるのに加え、ドメイン依存のヒューリスティクスが必要になるため、データベースを用意することは容易でない。また、前述のようなコマンドを得るためには少なくともユーザは「マージン」のような基本概念については知っている必要があり、「領域を右に寄せる」のような表現からコマンドを生成することはできない。展開ヘルプの手法はドメイン知識を利用してインテリジェントにコマンドを生成することはできないが、データの用意が簡単であり、エラーに対して寛容であるという特徴がある。

近年、IBM の Watson システム☆や Wolfram Alpha☆☆のように、複雑な人工知能テクニックを駆使することによって自然言語による質問文から解答を得るシステムが実用になってきている。これらのシステムは、開発者が設定した領域の問題をうまく解くことができるが、ヘルプが必要な任意の領域で適用できるわけではないし、普通のユーザがデータベースを用意したり回答をコントロールしたりすることができない。任意の問題に関して柔軟なヘルプを用意するためには展開ヘルプのようなシンプルでかつ集合知を利用しやすい手法が有効だと考えられる。

☆ http://www-03.ibm.com/innovation/us/watson/
☆☆ http://www.wolframalpha.com/

5. 結　　論

正規表現で表現したヘルプ文字列をリアルタイムに展開すると同時にユーザから与えられたキーワードで曖昧検索を行なう展開ヘルプシステムを提案した。展開ヘルプはヘルプデータの作成が容易であり、様々なシステムの機能を検索すると同時に実行する柔軟な枠組みとして有望である。

参　考　文　献

1) Sylvain Delisle and Bernard Moulin. User interfaces and help systems: from helplessness to intelligent assistance. *Artif. Intell. Rev.*, 18:117–157, October 2002.
2) G.W. Furnas, T.K. Landauer, L.M. Gomez, and S.T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30:964–971, November 1987.
3) Greg Little and Robert C. Miller. Translating keyword commands into executable code. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, pages 135–144, New York, NY, USA, 2006. ACM.
4) Sun Wu and Udi Manber. Fast text searching: allowing errors. *Communications of the ACM*, 35:83–91, October 1992.