JAPLA 研究会資料 2015/5/16

日本語のあいまい検索－J 正規表現プログラミングの適用 III

西川　利男

はじめに－あいまい検索との出会い

このところ植物に凝っている。道端に生えている雑草に、さて何という名の植物なのだろうかと思いをはせる。牧野植物図鑑をはじめとしていろいろ文献はとり揃えているが役にたっていない。大よその名前は分かるものの、正確な植物名になかなかたどり付けないのである。

例をあげると、家の近くの道端に、ひょろひょろと長い茎の先に赤い花をつける雑草をみかける。ケシの仲間らしいが、正確にはナガミノヒナゲシ（またはナガミヒナゲシ）という植物である。

日本語の植物名はどうして前に修飾句が付くのだろう。ここに、日本語の名前からの検索の難しさがある。植物図鑑の分類表や索引も役に立たない。つまり、通常の厳密な一致を探す検索とは違って、あいまい検索という課題に出っくわした。

１．日本語のあいまい検索とは

文字列の厳密一致の検索はいろいろなプログラミング言語で普通にサポートされている。しかしながら、上のような意味での日本語のあいまい検索を、コンピュータで行うことは、かなりめんどうである。たとえば、先の例で問題点を見てみよう。

ケシーーー＞ナガミノヒナゲシ

あいまい検索では、キーワードが名前の一部として含まれていればそれで検索できた、とする。つまり、上の例では、ケシは音便のためにゲシとなり、かつ、語の前に修飾句がついている。このような日本語の融通無碍の特質のために、これをコンピュータで扱うのは難しい。形容詞など修飾句が常に後ろに付くフランス語などではこんな悩みは生じないであろう。もちろん、植物の学名(papaver dubium)、ラテン語でも同様である。

２．J の正規表現プログラミング

正規表現（Regular Expression）とはPerlをはじめとして、Jはもちろん最近では多くのプログラミング言語でサポートされている共通の書式仕様である。

なお、Jの正規表現プログラミングについては、これを利用した、Jによる数式処理として以前発表したことがある。[1]

[1] 西川利男、「Jの正規表現プログラミングⅡ　数式処理システムへの利用」
APL/Jシンポジウム　2004/12/11

正規表現の考え方の基本はパターンマッチによる文字列の検出とその操作である。

Jの正規表現を用いたプログラミングは次のように行われる。くわしい仕様は最後に付録としてあげてある

まず、つぎのようにして正規表現を可能にする。

```
require ‘regex’
```

左引数に正規表現を、右引数に名詞を置き、つぎのJのシステム動詞で操作する。

rxmatches	マッチした文字列のインデックスを返す
rxall	マッチした文字列を取り出す
tryall	マッチした文字列の位置を示す

正規表現の「こてしらべ」としていくつかの操作をあげてみよう。

・左引数の文字列をパターンとして、右引数のマッチした位置にしるしをつける。

```
   'pqr' tryall 'abc 123+45 pqr **xyz***'
abc 123+45 pqr **xyz***
           ^^^
```

・左引数の文字列をパターンとして、右引数からマッチした文字列を取り出す。

```
   'pqr' rxall 'abc 123+45 pqr **xyz***'
+---+
|pqr|
+---+
```

・左引数に正規表現を使って、右引数からパターンがマッチした文字列を取り出す。

```
   '[[:digit:]]+' rxall 'abc 123+45 pqr **xyz***'
+---+--+
|123|45|
+---+--+
```

・J正規表現では、幸いに２バイトの日本語文字でも、上の操作は可能である。

```
   'に+' tryall 'にほんごににたことば'
にほんごににたことば
```

^^　　^^^^

このようなＪ正規表現を適用して、日本語のあいまい検索をプログラミングしてみた。

３．日本語あいまい検索の実際－植物名の検索への適用

Ｊの regex ライブラリに定義されている正規表現のシステム動詞を元として、つぎのような検索取り出しの動詞 trytake を定義した。

```
TAKEDA0 =: 'abc pqr xyz'
TAKEDA =: 'abc pqr xyz';'bc';' pqr ';'ab pqr'
NB. e.g. 'pqr' trytake >TAKEDA  => return results
NB.      'pqrs' trytake >TAKEDA => return null
trytake =: dyad define              NB. programmed 2015/4/27 by T.N.
n00 =: x. rxmatches y.              NB. if. no match,
if. 0 = $n00 do. return. end.       NB.   then return.
if. 0 = 1 { $n00 do. return. end.   NB. revised 2015/5/5
n0 =. 0 >. x. rxmatches y.
nn =: {. ″(2) {. ″(2) n0            NB. lower 2 to dim array
tn =. (0&<)″(1) {: ″(1) nn
tn # y.
)
```

```
   load'h:¥j402¥user¥nihongo_regex.ijs'
```

実行した結果はつぎのようになる。

```
 'ケシ' trytake 'ヒナゲシ'

 'ゲシ' trytake 'ヒナゲシ'
ヒナゲシ
```

つぎのようにいずれか一方にマッチするものも取り出される。

```
   '(ケ|ゲ)シ' trytake 'ヒナゲシ'
ヒナゲシ
```

次に、日本語の音便の扱いについて見ていこう。日本語文字は２バイト文字で（今の場合はシフト JIS）あらわしているので、次のようになる。

まず、日本語の２バイト文字の値は、動詞 val により求められる。

```
  val 'ケ'
131 80
```

これから、次のようにして清音から濁音へ文字の値は変換される。

```
  (0, 1) + val 'ケ'
131 81
```

文字の値から文字に戻せば、清音から濁音へ変換される。

```
  chr (0, 1) + val 'ケ'
ゲ
```

あいまい検索のために、つぎのようないくつかの動詞を定義した。

```
NB. 日本語カタカナ文字列を２バイトずつボックス化する
to_box =: 3 : 0
<"(1) ((-:@#y.), 2) $ y.
)
NB. 清音から濁音へ－日本語カタカナ１文字だけ変換
tseda =: 3 : 0
t =. ({: @ val) y.
if. (1 = (t >: 74) *. (t <: 122)) *. (0 = 2|t)
  do. yy =. chr (0, 1) + val y.
      '(', y. ,'|', yy, ')'
  else. y.
end.
)
NB. 音便－清音は(清音|濁音)、濁音はそのまま
NB. 複数文字列をボックス化して変換、文字列に戻す
onbin =: 3 : 0
; tseda L:0 to_box y.
)
NB. 植物データベース PLANT からあいまい検索する
search =: 3 : 0
(onbin y.) trytake > PLANT
)
```

実行のようすはつぎのとおりである。

```
   val 'カガ'
131 74 131 75
   tseda 'ガ'
ガ
   tseda 'カ'
(カ|ガ)
   onbin 'カガ'
(カ|ガ)ガ
   onbin 'ケシ'
(ケ|ゲ)(シ|ジ)
```

小さな植物名のリストPLANTを作って実験してみる。

```
  > PLANT
ナガミノヒナゲシ
ナガミヒナゲシ
オニノケシ
ケシ
ノゲシ
ハルノノゲシ
ハルジオン
ヒメジョオン
ヌスビトハギ
シナガワハギ

  (onbin 'ケシ') trytake > PLANT
ナガミノヒナゲシ
ナガミヒナゲシ
オニノケシ
ケシ
ノゲシ
ハルノノゲシ
```

最終的にはデータベースPLANT に対して、searchコマンドにより検索する。

```
  search 'ケシ'
ナガミノヒナゲシ
ナガミヒナゲシ
オニノケシ
ケシ
ノゲシ
ハルノノゲシ

  search 'ハギ'
ヌスビトハギ
シナガワハギ
```

Jプログラミングリスト

```
NB. Regular Expression on J

require 'system¥main¥regex.ijs'

NB. 植物名の検索への適用
NB.   2015/4/27

PLANT =: 'ナガミノヒナゲシ';'オニノケシ';'ケシ';'ノゲシ';'ハルノノゲシ';'ハルジオン';'ヒメジョオン'

NB.   'x|y' tryall 'axb'
NB. axb
NB.  ^
NB.   'x|y' tryall 'ayb'
NB. ayb
NB.  ^

NB.   '(ケ|ゲ)シ' tryall 'ヒナゲシ'
NB. ヒナゲシ
NB.     ^^^^
NB.   '(ケ|ゲ)シ' tryall 'ナガミヒナケシ'
NB. ナガミヒナケシ
NB.           ^^^^
NB.   '(ケ|ゲ)シ' tryall 'ナガミヒナゲシ'
NB. ナガミヒナゲシ
NB.           ^^^^



NB. File Manipulation Utilities ============================
require 'files'
dir =: fdir

wr=: 1!:2&2
rd=: 1!:1
```

```
each=: &. >

deb=: #~ (+. 1&|.@(> </¥))@(' '&~:)

h=:  '0123456789ABCDEF'
dfh=: 16&#. @ (h&i.) f.
hfd=: dfh ^:_1 f.

NB. hex  (adverb)
NB. e.g. 'FF' + hex '8'
hex=: &. dfh

val=: a. & i.
chr=: val ^:_1
hdump=: }:@,@(,"1' '"_)@hfd@val f.

NB. Ｊの正規表現プログラミングの要点（文献[1] より再録）
NB. 正規表現によるパターンの指定
NB. ・個々の文字              'abc', '3.14', 'にほんご'（２バイト文字も可能）
NB. ・特殊文字          '¥.'⇔ピリオドそのもの, '¥('⇔かっこ
NB. ・すべての文字      .  (ピリオド)
NB. ・いずれか１文字   '[a-z]', '[A-Z]', '[0-9]'
NB.                    '[aeiou]'          母音文字
NB.                  '[[:alpha:]]'     英文字
NB.                  '[[:alnum:]]'     英数文字
NB.                  '[[:digit:]]'     数字
NB.              '[[:space:]]' スペース
NB.                  '[[:punct:]]'     区切り文字
NB.                  '[[:cntrl:]]'     制御文字
NB. ・選択        'a|b'   aまたはb
NB. ・除いた文字          '[^aeiou]'        母音を除いた文字
NB. ・繰り返し ＊  ０回～     'ab*c'         ac, abc, abbc, abbbc にマッチ
NB.              +  １回～     'ab+c'         abc, abbc, abbbc にマッチ
NB.              ？  ０回/１回 'ab?c'         ac, abc にマッチ
NB. ・文字列の繰り返し         かっこ( )でくくる
```

```
NB. ・取り出し反復              かっこ( )でくくる
NB. 正規表現の基本操作関数
NB. ・マッチした文字列にしるしをつける.
NB.   '[[:digit:]]+' tryall 'abc234def43'
NB. abc234def43

NB.   ,<"(1)'[[:digit:]]+' rxmatches 'abc234def43'
NB. +---+---+
NB. |3 3|9 2|
NB. +---+---+

NB.    ^^^   ^^
NB. ・マッチした文字列を取り出す.
NB.   '[[:digit:]]+' rxall 'abc234def43'
NB. +---+--+
NB. |234|43|
NB. +---+--+

try =: dyad define     NB. try original function
y. ,: '^' #~ 0 >. {. x. rxmatch y.
)
NB. tryall by T.N. 2004/7/19 modified from try function
NB. f.g.
NB.   '[aeiou]+' tryall 'These reveal various examples'
NB. These reveal various examples
NB.   ^ ^  ^ ^^   ^ ^^^  ^ ^   ^
NB.   'に+' tryall 'にほんごににたことば'
NB. にほんごににたことば
NB. ^^      ^^^^
orsp =: ('^'"_)`(' '"_)@.=      NB. eq=> ' ', nq=>'^'
tryall =: dyad define
n0 =: 0 >. x. rxmatches y.      NB. take only positive, if _1(=no match) => 0
n1 =: {. "(2) n0                NB. convert to 2 dim array
n2 =: n1 #' ^'                  NB. convert to 'sp^' pattern
n3 =: orsp/n2
y. ,: n3
```

```
)

TAKEDA =: 'abc pqr xyz'
TAKEDA1 =: 'abc pqr xyz';'bc';' pqr ';'ab pqr'

trytake =: dyad define
n0 =. 0 >. x. rxmatches y.
nn =: {. "(2) {. "(2) n0        NB. lower 2 to dim array
tn =. (0&<)"(1) {: "(1) nn
tn # y.
)



NB.
NB. ・マッチした文字列を置換する.
NB.   ('[[:digit:]]+';'***') rxrplc 'abc234def43'
NB. abc***def***
NB. ・マッチした文字列に操作を行う.
NB.   '[[:alpha:]]+' |. rxapply 'abc234def43'
NB. cba234fed43

NB.   'a(b[aeiou])c' try 'x abec x'
NB. x abec x
NB.   ^^^^

NB.   'a(b[aeiou])c' rxmatch 'x abec x'
NB. 2 4
NB. 3 2
NB. First row:   2 4  is the 'abec' string
NB. Second row:  3 2  is the 'be' substring
```