J-OpenGL によるルービック・キューブの３Ｄグラフィックス－３
－Ｊの正規表現によるルービック操作のイディオム入力と実行－

西川　利男

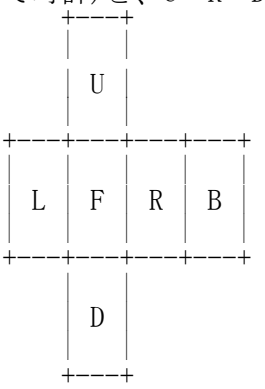
Ｊによるルービック・キューブをこれまで３回にわたって報告してきた [1][2][3]。最初はルービック・キューブが話題になったかなり早い時点で、中野嘉弘先生との共著であるが、グラフィックスといっても展開図であった[1]。昨年から再度取り上げ、ルービック・キューブの単なる 3D グラフィックスではなく、動きを伴う一種のアニメーションとした。これを J-OpenGL を用いてルービック操作のシミュレーション・システムとして構築した[2]。さらにルービック操作を群論の置換群として扱い、汎用性を持たせるよう整備、再構築している[3]。

基本のルービック操作は、ルービック面に対応した正逆 12 種類の回転と中間エッジの正逆 6 種類の回転がある。そして実際にはこれらを組み合わせて、いろいろなイディオム的な操作が使われる。これらは相当ステップ数も多く、個別に入力するのも大変である。このようなルービック操作を数式のようにプログラムとして入力する方式を、Ｊの正規表現を利用して作った。同時にユーザ・インターフェースを充実させ、より使い易いシステムへと改善を行っている。今回はその現状を報告したい。

１．ルービック操作とそのイディオム入力

ルービック操作は前回[3]と同じに、例えば、上面での反時計まわりの回転を u、時計まわりの回転を U、…　とする。「宝島社」攻略本[4]では U, U’ のようにしている。

また、中間の辺を通る回転は U→F→D→B では m(R 面から見て反時計), M(R 面から見て時計)と、U→R→D→B では s, S と、L→F→R→B では e, E とそれぞれ名付けた。



	当プログラム, Joyner	宝島社 攻略本[4]
反時計まわり	u(U^{-1}), l, f, r, b, d	U’, L’, F’, R’, B’, D’
	m, s, e　M_R^{-1}	m, s, e
時計まわり	U　, L, F, R, B, D	U, L, F, R, B, D
	M, S, E　M_R	M, S, E

上のようにルービック操作を文字で指示するときは、その連続操作は一種の文字列である。したがって、長い文字列の反復などを容易に能率的に行うこが必要になってくる。そのための文字列処理にＪの正規表現を利用して、より便利に入力が行えるようにした。

２．Ｊの正規表現とは

Ｊの正規表現については前にも紹介したが、その要点を採録する。

Ｊの正規表現プログラミングの要点（文献[5] より再録）

Ｊの正規表現ライブラリの取り込み

```
require 'system¥main¥regex.ijs'
```

正規表現によるパターンの指定

・個々の文字　　'abc', '3.14', 'にほんご'（２バイト文字も可能）
・特殊文字　　'¥.'⇔ピリオドそのもの, '¥('⇔かっこ
・すべての文字　　.　（ピリオド）
・いずれか１文字　'[a-z]', '[A-Z]', '[0-9]'
　　'[aeiou]'　　母音文字
　　'[[:alpha:]]'　　英文字
　　'[[:alnum:]]'　　英数文字
　　'[[:digit:]]'　　数字
　　'[[:space:]]'　　スペース
　　'[[:punct:]]'　　区切り文字
　　'[[:cntrl:]]'　　制御文字
・除いた文字　　'[^aeiou]'　　母音を除いた文字
・繰り返し ＊　０回～　　'ab*c'　　ac, abc, abbc, abbbc にマッチ
　　＋　１回～　　'ab+c'　　abc, abbc, abbbc にマッチ
　　？　０回/１回 'ab?c'　　ac, abc にマッチ
・文字列の繰り返し　　かっこ( )でくくる
・取り出し反復　　かっこ( )でくくる

正規表現の基本操作関数

・マッチした文字列にしるしをつける.

```
   '[[:digit:]]+' tryall 'abc234def43'
abc234def43
   ^^^   ^^
```

・マッチした文字列を取り出す.

```
   '[[:digit:]]+' rxall 'abc234def43'
+---+--+
|234|43|
+---+--+
```

・マッチした文字列を置換する.

```
   ('[[:digit:]]+';'***') rxrplc 'abc234def43'
abc***def***
```

・マッチした文字列に操作を行う.

```
   '[[:alpha:]]+' |. rxapply 'abc234def43'
cba234fed43
```

３．Ｊの正規表現を利用したルービック操作の文字列処理

例として、次のルービック・イディオムをとりあげる。

$F^2\ L^2\ U^2\ (F^2\ L^2)^3\ U^2\ L^2\ F^2$

英文字のそれぞれは、ルービック操作を示し、べき乗はその文字の繰り返しである。
また、かっこのべき乗はかっこ部分の繰り返しである。

キーボード上では次のように入力して操作を行いたい。

```
   F2 L2 U2 (F2 L2)3 U2 L2 F2
```

すると、操作は以下のようになされる。

```
   FF LL UU FF LL FF LL FF LL UU LL FF
```

これにしたがって、ルービック操作を実際に行うと、U(上面)の４つのエッジ・キューブの間で、十字交換が行われる。

このように、ルービックの入力操作をべき乗、かっこを用いて行えるようにしたい。

４．Ｊの正規表現プログラム

プログラム calc は基本として次の２つから成る。

・べき乗を積の繰り返しへ…… pow2mul
・かっこをはずす…… par2mul

４．１ べき乗を積の繰り返しへ

ここでは、その基本部分を説明する。

```
   PA =: 'b2c3'
  ]PB =: '[[:alpha:]]+[[:digit:]]' rxall PA
+--+--+
|b2|c3|
+--+--+
  ]P1 =. _1{ L:0 PB
+-+-+
|2|3|
+-+-+
  ]P2 =. _2{ L:0 PB
+-+-+
|b|c|
+-+-+
   P1 p2m P2
bbccc
```

ここで、サブプログラム p2m は P2 の文字を P1 の回数だけ繰り返す。

```
  ]PA =: 'ab2c3'
ab2c3
```

のときは、サブプログラム ad1 により'a'を'a1'と変形した後、正規表現処理を行う。

```
  PB
+--+--+--+
|a1|b2|c3|
+--+--+--+
   P1 p2m P2
abbccc
```

４．２ かっこをはずす

次の例でやってみる。

```
   par2mul 'ab3(xy2z)3d2'
ab3xy2zxy2zxy2zd2
```

プログラムは以下のようである。

```
par2mul =: 3 : 0
P0 =. '.*(¥(.+¥)).*' rxmatches y.  NB. $P=1 2 2
P2 =. ,{:"2 P0                     NB. take 2nd row of P
P3 =. ({. P2) + i. {: P2
Q0 =. P3 { y.
A =. ({. P2) {. y.
B =. (>: >:{: P3) }. y.
C =. (>: {: P3) { y.
Q1 =. }. }: Q0
Q =. ,(". C)#,: Q1
A, Q, B
)
```

プログラムのポイントはつぎの正規表現のかっこ文字¥(と¥)で囲まれた文字列のインデックスの取り出しである。

```
   P0 =. '.*(¥(.+¥)).*' rxmatches y.
   P0
0 12    全体マッチした位置
3  6    副次マッチした位置
   P2
3 6    副次マッチした位置だけ
   P3
3 4 5 6 7 8  かっこ部分のインデックス
   Q0
(xy2z)       かっこ部分の文字列
   A
ab3          かっこより前の文字列
   B
d2           かっこより後の文字列
   C
3            かっこ部分のべき乗の値
   Q1
xy2z         かっこの内容の文字列
   Q
xy2zxy2zxy2z かっこの内容にべき乗を実行
   A, Q, B
ab3xy2zxy2zxy2zd2 かっこより前、かっこの内容にべき乗した結果、かっこより後を
                  連結して、最終の文字列を得る。
```

５．ルービック操作のイディオム入力と実行のようす

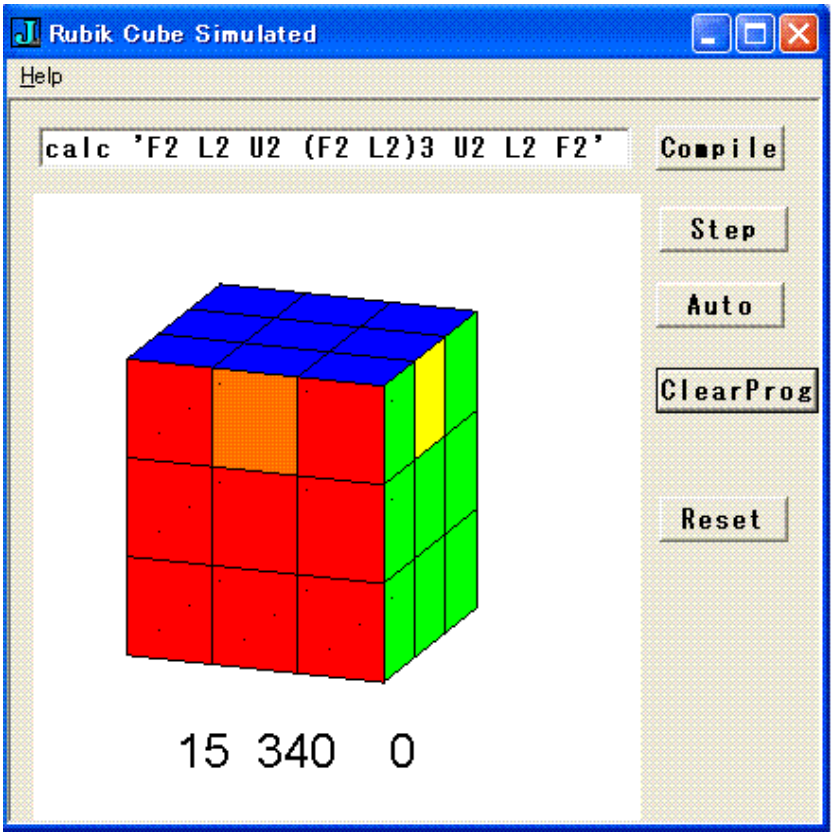
先にあげたU(上面)エッジ・キューブの十字交換の入力と実行のようすを示す。

入力窓にcalcにルービックイディオム操作を文字列で入力して、エンターキーを押してから、Compileボタンを押す。すると入力窓は、

FF LL UU FF LL FF LL FF LL UU LL FF

と表示が変わる。さらにエンターキーを押して、Autoボタンを押すと以下のようにルービック操作が行われて、十字交換された画面に変わり表示される。

なお、Stepボタンでは、途中経過のようすが次々と表示される。



文献

[1] 西川利男、中野嘉弘、林　雄二「J言語によるウィンドウズ・プログラミングールービック・キューブのシミュレーション」　北海道情報大学紀要、第10巻第1号, p.219-240 (1998).

[2] 西川利男「J-OpenGLによるルービック・キューブの３Ｄグラフィックス」
JAPLA研究会資料 2011/10/22

[3] 西川利男「J-OpenGLによるルービック・キューブの3Ｄグラフィックスー２ーー群論（置換、巡回置換）によるルービック操作とプログラムー」JAPLA研究会資料 2012/1/28

[4] 「頭を鍛えるルービックキューブ完全解析！」宝島社(2007).

[5] 西川利男「Ｊの正規表現プログラミングーⅠ，正規表現とはーPerlと比較しつつ，Ｊのボックス表示の文字化け解消への利用」Ｊ言語研究会資料2004/9/25.

[6] 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

プログラム・リスト

```
NB. べき乗を積の繰り返しへ  ========================================
require 'system¥main¥regex.ijs'
NB.
NB.    calc '(ER)3R(eR)3R'   2012/1/20
NB. ERERERReReReRR

calc =: 3 : 0
y =. y. -. ' '
while. '(' e. y do. y =. par2mul y end.
pow2mul y
)

NB.    calc 'Fr2'
NB.    calc 'ab3(xy2z)3d2'
NB. abbbxyyzxyyzxyyzdd

NB. power to multiple - revised 2011/12/13
NB.   pow2mul 'ab3xy2zxy2zxy2zd2'
NB. abbbxyyzxyyzxyyzdd

pow2mul =: 3 : 0
NB. single alpha to alpha + 1 - revised 2011/12/13
p =. y. , ({: y.)         NB. revised 12/14
PP =. 2 <¥ p
PPP =. ; ad1 L:0 PP
NB. 'ab2c3' => 'abbccc'
P =. '[[:alpha:]]+[[:digit:]]' rxall PPP
P1 =. _1{ L:0 P
P2 =. _2{ L:0 P
P1 p2m P2
)

asc =: a.&i.

alp =: (64&< * <&123)@asc
num =: (47&< * <&58)@asc

ad1 =: 3 : 0
'A1 B1' =. y.
AD1 =. y.
if. alp B1 do. AD1 =. A1, '1' end.
if. num A1 do. AD1 =. '' end.
AD1
)
```

```
p2m =: 3 : 0
:
n =. #y.
i =. 0
PM =. ''
while. i < n
  do.
    PM =. PM , (". >i{x.) # (>i{y.)
    i =. i + 1
  end.
PM
)

NB. かっこをはずす 2011/12/14 ============================================
NB.     par2mul 'ab3(xy2z)3d2'
NB. ab3xy2zxy2zxy2zd2
par2mul =: 3 : 0
P0 =. '.*(¥(.+¥)).*' rxmatches y.  NB. $P=1 2 2
P2 =. ,{:"2 P0                     NB. take 2nd row of P
P3 =. ({. P2) + i. {: P2
Q0 =. P3 { y.
A =. ({. P2) {. y.
B =. (>: >:{: P3) }. y.
C =. (>: {: P3) { y.
Q1 =. }. }: Q0
Q =. ,(". C)#,: Q1
A, Q, B
)

NB.     unpar 'ab3(xy2z)3d2(p2q)2e3'
NB. ab3xy2zxy2zxy2zd2p2qp2qe3
unpar =: 3 : 0
np =. +/ '(' = y.
i =. 0
R =. y.
while. i < np
  do.
    R =. par2mul R
    i =. i + 1
  end.
R
)
val=: a. & i.
to_small =: (32&+) &. val
```

```
to_large =: (-&32) &. val
```