PCで解く知恵の輪

Puzzling Out Puzzle Links Using PCs

ネットワーク情報学部　石鎚 英也
School of Network and Information　Hideya ISHIZUCHI

Keywords: puzzle links, Chinese ring, Prolog, difference equation, recursiveness, Gray code.

1. はじめに

ふと立ち寄った100円ショップで知恵の輪を見つけた。キャストパズルと呼ばれるものは高価で二の足を踏む所だが、これならリスク(価格に対して面白味が少ない)はほとんどないだろう。「前に自分のおもちゃを買ったのは一体何(十)年前だっけ」と思いながらいくつか買って帰った。

一言で知恵の輪といってもずいぶん種類があるようで、金属の円盤にたくさんの穴が開いているものや迷路のようなものもある。しかし、やはり“輪”でできているものが知恵の輪らしくて良い[1]。

中には相当難しいものもあって、この手のパズルが余り得意ではない著者にとっては、冬休みの暇つぶしのつもりがストレスになりかねない…ということで、PCに解かせてみようかと少し調べてみた。

2. 目標

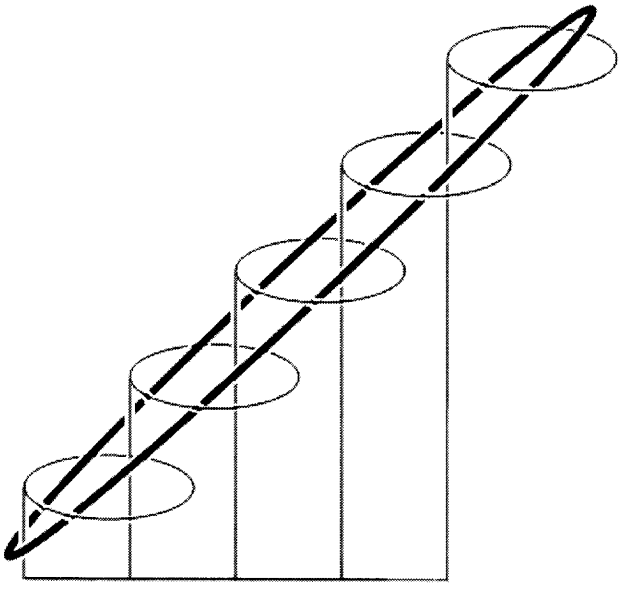


図1　五連環

輪でできている知恵の輪にも色々あるが、ここでは、図1のような知恵の輪[2]を想定して、PCを使ってできるだけ簡単に解くことを試みたい。

図は、“九連環”(あるいはチャイニーズリング[3])と呼ばれる古来中国から伝わる知恵の輪と本質的に同じものである(これにちなんで、図の知恵の輪を“五連環”と呼ぶことにする)。

図の細い線は針金で、形を変えられないものとする。太線は閉じた紐を示し、形は自由に変えられ、また十分に長いものとする[4](長い輪ゴムのようなものを想定しても良い)。

3. トポロジカルな変形

これが原理的に解けることは、針金の方を連続的に変形することで分かるであろう。つまり、紐ではなく針金の方を自由に変形できるものとすると、紐の右先端から針金の輪を右上方に外し、紐の内部を上から下にくぐらせるという操作を右の輪から左の輪にかけて順次行えばよい。

ただし、解があると分かること(解の存在性)と解を求める手続きが分かる(あるいは、解を“効率的に”求める手続きが分かる)こととは別物[5]であるように、(前提知識なしに)五連環を具体的に解くことは、それほど容易ではない[6]。

なお、より複雑な構造の知恵の輪ではあるが、文献[3]のウェブページには、JAVAアプレットによるこうした変形を示すアニメーションがあるので参照されると良い。

---

[1] 買ったものの中に、“馬蹄形”の知恵の輪と呼ばれるものが(形は違っていたが)あった。非常にシンプルながら(というより、シンプルであるが故に)気にいった。

[2] 輪と軸からなる知恵の輪の基本的な構造について文献[5]を参考にした。

[3] チャイニーズリングは、もっとも古い種類の知恵の輪と考えられている。『戦国策』には、「秦の昭王が斉国に玉連環を贈った」という記述が出てくる。確証はないがこの「玉連環」が、「九連環」と同種のものであるといわれている(文献[2]より引用)。また、文献[6]も参照。

[4] 実際の九連環では太線部分は紐ではなく針金であって、自由に曲げることはできない。

[5] 「解は存在します。しかしその値は分かりません」という面白いタイトルの森先生の解説記事(文献[8])がある。代数の話であるが、暗号の基礎の1つ「(大きな素数による)素因数分解の困難性(計算量の壁)」もその1例であろう。

[6] 十三連環やそれ以上のものもあるそうで、五連環が物足りないというみなさんでも(少なくとも)十分な暇潰しにはなるだろう。

4. モデル化

PCで解くためには、知恵の輪をモデル化[7]する必要があるが、一体どうすればよいのだろうか。

以下では、長崎総合科学大学基礎教育センターのウェブページ(文献[3]参照)に記されている考え方を用いてみよう。

まず、場所を特定するための名前を五連環のパーツにつけ、それを用いて、操作（紐の動き）を定義する。なお、ここでは、プログラミング上の便宜等のため、同ウェブページの記法とは異なる点がある。

【パーツの定義】

五連環のパーツを、図2のような記号を用いて識別する。つまり、輪は左から①, ②, …と表し、輪から下に伸びる軸の下部を左から[1], [2], …と表す。軸の上部は、左から[1, 2], [2, 3], …のように表現する。一般に[N, N+1]はN番目の輪とN+1番目の輪の間の軸の部分であり、その下の部分が[N+1]である。

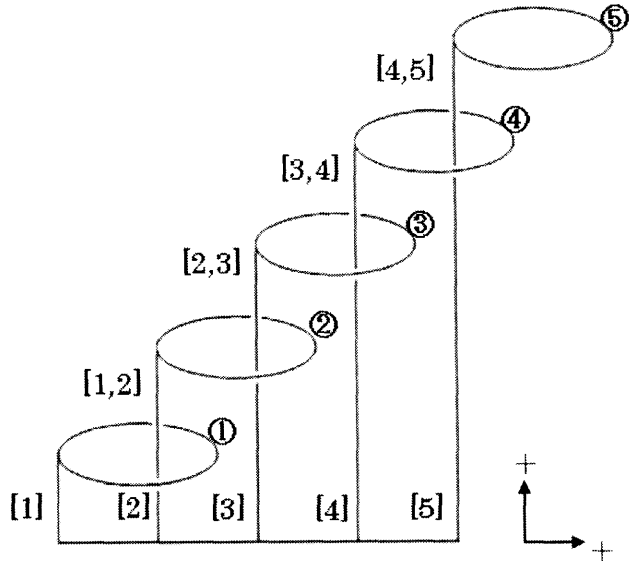


図2　パーツの名称

4-1. 紐の動き

解を記述するためには、紐の動きを表現する必要がある。「複雑なものは単純化すべし」という原則に従って、より簡単な図3のケースを考えてみよう。これなら容易に解ける。つまり、(1) 紐の右端を輪①の中を下から上に通し、(2) 輪②の外を下から上に回り込ませ、(3) 輪①の中を上から下に通せば良い[8]。

[7] モデル化（モデリング）では、現実世界を他の世界の言葉（図表・模型・数学・プログラム等）に翻訳（定式化）し、その世界の方法で操作し、得られた知見を現実世界で解釈する。坂本先生の解説（文献［4］）を参照されたい。

[8] あるいは、紐の左端を動かすとすると、(1) ①の中を下から上に通し、(2) ②の外を上から下に回りこませ、(3) ①の中を上から下に通せば良い。

このように、紐の動きは、(a) いずれかの輪の中を下から上（あるいは上から下）に通すこと、(b) いずれかの輪の外を下から上（あるいは上から下）に回り込ませることに分けられるだろう。

このことから紐の動きを以下のように定義しよう。

【紐の具体的な動き[9]】

- *pp* (*N*)：輪Ⓝの中を下から上(＋方向とする)に通す。
- *pm* (*N*)：輪Ⓝの中を上から下(－方向とする)に通す。
- *tp* (*N*)：輪Ⓝの外を下から上（＋方向とする）に回り込ませる。
- *tm* (*N*)：輪Ⓝの外を上から下（－方向とする）に回り込ませる。

なお、図2や図3の垂直方向の矢印は、下から上への動きが正の方向であることを意味している。

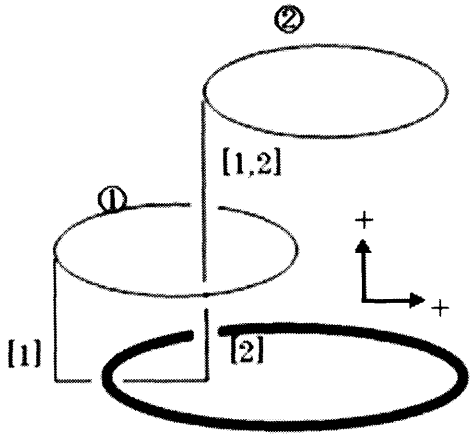


図3　紐の動き

この記法に従うと、図3の解（紐の右端）は以下のように記述される[10]：*pp* (1) → *tp* (2) → *pm* (1)

4-2. 解法の定式化

このように、*pp(N)*, *pm(N)*, *tp(N)*, *tm(N)*の系列によって、解（針金から分離されるまでの一連の紐の動き）を表現できれば良いことになる。前述のウェブページでは、“紐の仮想的な動き”という考え方を使って定式化を行っている。

【紐の仮想的な動き】（図4参照）

- *p (N)*：軸 [N] を左から右(＋方向とする)に通過する。
- *m (N)*：軸 [N] を右から左(－方向とする)に通過する。
- *p* (*N*, *N*＋1)：軸 [N, N+1] を左から右(＋方向とする)に通過する。

[9] 記号の最初の文字は動きの種類（pはpass, tはturnのつもり）を示し、後の文字は動きの方向（pはplus, mはminusのつもり）を示す。

[10] 紐の左端を動かすケースだと、pp(1)→tm(2)→pm(1)である。

・$m(N, N+1)$：軸 [N, N+1] を右から左（－方向とするに通過する。

図の下部は m (2)（「実線→波線」の変化）と p (2)（「波線→実線」の変化）を、上部は m (1,2)（「実線→波線」の変化）と p (1, 2)「波線→実線」の変化）
をそれぞれ示している。

なお、図2や図3の水平方向の矢印は、左から右への動きが正の方向であることを意味している。

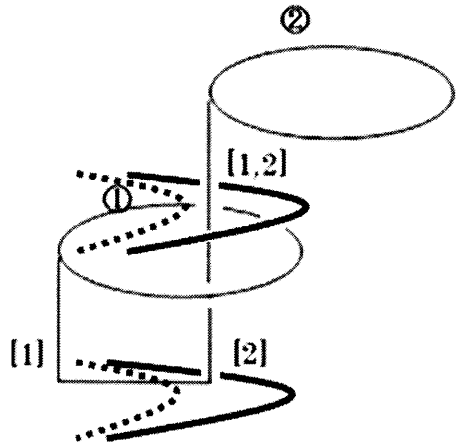


図4　紐の仮想的な動き

勿論、これらの操作は直接的に実現できるものではなく、紐の“仮想的”な動きであるが、図3（あるいは図4）では、以下の関係が成り立つことが分かるだろう。

・p (1,2) は tm (2)（輪②の外を上から下に回りこむ）により実現できる。
・m (1,2) は tp (2)（輪②の外を下から上に回りこむ）により実現できる。
・p (2) は pp (1) → p (1, 2) → pm (1)（輪①の中を下から上に通し、[1, 2] を仮想的に通し、輪①の中を上から下に通す）により仮想的に実現でき、さらに p (1,2) を tm (2) に置き換えることで、具体的に実現できる。
・m (2) についても同様である。

“紐の仮想的な動き”という概念のメリットは、それを使うことによって、非常に簡単に解を表現できることである。図3を解く場合だと、単に p(2)（紐の左端を動かす場合）あるいは m(2)（紐の右端を動かす場合）とすれば良いだけである。勿論、これは仮想的な解の表現であるが、上述の関係から具体化する（実際の操作に結び付ける）のも容易である。

以上のアイデアを一般化すると、次のような解法の“戦略”が得られる。

1. 仮想的な動きと具体的な動きを結びつける関係を明らかにする。
2. 解を仮想的に表現する（具体的な動きが含まれていてもかまわない）。
3. 1を使って、解の仮想的な表現を具体的な表現に変換する。

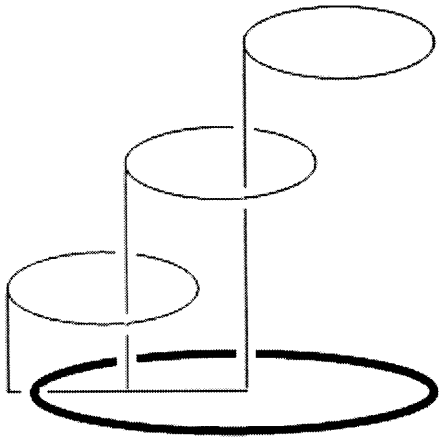


図5　簡単な例

この考え方を使って、図3を少しだけ複雑にした図5の知恵の輪を解いてみよう（パーツの名称は図2と同様である）。

● 関係

p (2, 3) ▷ tm (3)
m (2, 3) ▷ tp (3)

$N = 2, 3$ について
$p(N) \triangleright pp(N-1) \rightarrow p(N-1, N) \rightarrow pm(N-1)$
$m(N) \triangleright pp(N-1) \rightarrow m(N-1, N) \rightarrow pm(N-1)$

$N = 1$ について
$p(N, N+1) \triangleright p(N+1, N+2) \rightarrow tm(N+1) \rightarrow m(N+2)$
$m(N, N+1) \triangleright p(N+2) \rightarrow tp(N+1) \rightarrow m(N+1, N+2)$

上記の ▷ は、その左辺を実現するためには右辺を実現すれば良いことを意味する。

なお、上に見られるように、ある関係が成立すれば、系列の順序を逆順にし、その中に現れる（正負の）方向を下記のように逆に置き換えた関係についても成立することが理解されるだろう。そのような意味で、これらの関係は“双対的”である。

$p(N) \leftrightarrow m(N)$, $p(N, N+1) \leftrightarrow m(N, N+1)$
$pp(N) \leftrightarrow pm(N)$, $tp(N) \leftrightarrow tm(N)$

● 解の仮想的な表現

右端を動かすとすると、$m(3) \rightarrow m(2)$ が解であり、紐の左端を動かすとすると、$p(2) \rightarrow p(3)$ が解である。解の表現についても上記のような意味の双対性がある。

● 具体的な表現への変換

$m(3) \rightarrow m(2)$ を具体的に表現してみよう。$m(3)$ については、

$m(3)$ ▷ pp (2) → m(2, 3) → pm (2)
▷ pp (2) → tp (3) → pm (2) である。

$m(2)$ の式 $m(2) \triangleright pp\ (1) \rightarrow m(1,2) \rightarrow pm\ (1)$ の右辺で仮想的なのは $m(1,2)$ であり、関係
$m(1,2) \triangleright p\ (3) \rightarrow tp(2) \rightarrow m\ (2,3)$ と
$m(2,3) \triangleright tp\ (3)$ により、
$pp\ (1) \rightarrow p\ (3) \rightarrow tp\ (2) \rightarrow tp(3) \rightarrow pm\ (1)$ が得られ、
$p(3) \triangleright pp\ (2) \rightarrow p(2,3) \rightarrow pm\ (2)$ と
$p(2,3) \triangleright tm\ (3)$ により、
$pp\ (1) \rightarrow pp(2) \rightarrow tm\ (3) \rightarrow pm\ (2) \rightarrow tp\ (2) \rightarrow tp(3) \rightarrow pm\ (1)$
と具体化される。以上をまとめると、

$$m(3) \rightarrow m\ (2) \triangleright pp\ (2) \rightarrow tp(3) \rightarrow pm\ (2) \rightarrow pp\ (1) \rightarrow pp\ (2) \rightarrow tm(3) \rightarrow pm\ (2) \rightarrow tp\ (2) \rightarrow tp\ (3) \rightarrow pm(1) \quad \cdots(0)$$

が解として得られる。
$pp\ (2) \rightarrow tp(3) \rightarrow pm\ (2)$ で紐が [2] を取り巻く形になり、
$pp\ (1) \rightarrow pp\ (2) \rightarrow tm(3) \rightarrow pm\ (2)$
で、輪 ① の中を経て軸 [3] を取り巻く形になり、最後に $\rightarrow tp(2) \rightarrow tp(3) \rightarrow pm\ (1)$ で、紐の右端を反時計回りに大きく回転させて輪 ① を上から通すという様子が想像できるだろうか。

これまでのことから推察できるように、連環の輪が1つ加わるだけで、定式化された関係を人手に頼って展開することはとても煩雑な作業になる。

5. プログラミング

ここでは、当初の目的であった五連環の定式化とそのプログラミングについて簡単に触れよう。

図5のときと同様に（やや一般化して）考えれば、以下のような差分方程式（漸化式）が得られる（n=5のケースが五連環である[11]）：

● 関係

$$p(n-1,n) \triangleright tm(n)$$
$$m(n-1,n) \triangleright tp(n) \quad \cdots(1)$$

$N = 2,\cdots,n$ について

$$p(N) \triangleright pp(N-1) \rightarrow p(N-1,N) \rightarrow pm(N-1)$$
$$m(N) \triangleright pp(N-1) \rightarrow m(N-1,N) \rightarrow pm(N-1) \quad \cdots(2)$$

$N = 1,\cdots,n-2$ について

$$p(N,N+1) \triangleright p(N+1,N+2) \rightarrow tm(N+1) \rightarrow m(N+2)$$
$$m(N,N+1) \triangleright p(N+2) \rightarrow tp(N+1) \rightarrow m(N+1,N+2) \quad \cdots(3)$$

[11] 通常の数学的な記法の場合、添え字（や関数の引数）にはアルファベットの小文字を、添え字の最大値には大文字を使うのが普通で違和感を覚えるかも知れないが、後述のプログラム言語に合わせてここでは逆にしている。

● 解の仮想的な表現

これまで、図の太線を紐と表現してきたが、実際の五連環では紐ではなく、これも（細長い）針金でできている。従って、図1の“紐”は、実は、右端しか動かせないのである[12]。そこで、ここでは紐の右端を動かす解を求めることにしよう。これは、“紐の仮想的な動き”を用いれば簡単で、下記のようにすればよいだろう。

$$pm(5) \rightarrow m(4,5) \rightarrow pm(4) \rightarrow m(3,4) \rightarrow pm(3) \rightarrow m(2,3) \rightarrow pm(2) \rightarrow m(1,2) \rightarrow pm(1) \rightarrow p(2) \rightarrow p(3) \rightarrow p(4) \rightarrow p(5) \rightarrow tp(5) \quad \cdots(4)$$

● コードの例

基本的には、差分方程式(1)～(3)を使って、仮想的動きを含む(4)式を具体的動きのみで表現すれば良い。差分方程式は、関数の再帰的な定義としてプログラムに実装される。実際、ハノイの塔[13]の解法に典型的に見られるように、プログラムでパズルを解く際に“再帰（recursiveness）[14]”は良く用いられるようである。

どんなプログラミング言語を使っても良いが、この種の問題を比較的自然に表現できるため、Prolog[15]を使ってみた。ここでは、フリーの“SWI-Prolog[16]”というソフトを利用した（起動画面を付録に示す）。以下（図6）にコードの例を示すが、洗練化されたものでないことをお断りしておく。

概要は以下のとおりである[17]。

(a) 述語 “solve” の最初の引数は、これまでの定式化の ▷ の左辺に、第2引数は ▷ の右辺に相当する。ただし、動きの系列は、矢印の代わりにカンマで区切られたリスト[18]として表現する。

(b) 最初の2文は(1)式（および n=5）に相当する。

(c) 次の4文は、pp(N), pm(N), tp(W), tm(N)の具体化はそれ自身（のリスト）であることを意味する。

(d) 次の2文[19]は(2)式に相当する。第1のものは以下を意味

[12] 勿論その細長い針金の中に輪を通すといったことは（物理的に無理のない範囲で）できる。

[13] 例えば、文献 [7] を参照。

[14] プログラムにおける再帰とループの関係について石原先生が解説されている（文献 [1] 参照）。そこで述べられているハノイの塔の2つのプログラム（再帰と非再帰）を比較すれば、再帰の有用性が実感できるだろう。

[15] Prologは宣言的言語で、CやJAVAなどの手続き的言語とはかなり異なる書法を用いる（「howではなくwhatを記述する」と言われることもある）。手続き的言語の関数は、Prologでは、論理学的な述語という言葉で呼ばれる。

[16] SWI-Prologのサイト（文献 [10]）からダウンロードできる。

[17] ここでは細かい文法等には触れないので、Prologになじみのない方は雰囲気だけ感じていただければよいと思う。詳しくは、マニュアルや解説（例えば、文献 [9]）を参照。

[18] 例えば、m(3)→m(2)という手順は、カギカッコの中にコンマで区切って要素（手順）を並べた〔m(3),m(2)〕というリストで表わされる。

[19] 紙面スペースの関係で、図6では、長い1文が折り返されて表示されている。

する：p(N)を具体化した表現が[pp(N1), X, pm(N1)]という“パターン”であるのは、N1がN-1であり、Xがp(N1, N)を具体化したものであるときである。最後の“!”はカットと呼ばれ、ここでは、パターンマッチングに失敗したときそれ以上の探索を行わないことを意味する。
(e) 次の2文は(3) 式に相当する。意味は(2)式の場合と同様である。
(f) 次の2文は、solveの第1引数がリスト（手順の系列）である場合、それぞれの要素（手順）の具体化を行い、結果をつなげた（append）ものが全体の具体化であることを意味する。なお、これだけでは、リスト内にリストが存在して煩雑になるので、組み込み述語“flatten”により不要なカギカッコを消して、平坦な構造に変更している。
(g) 最後の文では、solveの第1引数を入力して第2引数を計算し表示する述語を定義している。

```
solve(p(4,5),[tm(5)]).
solve(m(4,5),[tp(5)]).

solve(pp(N),[pp(N)]).
solve(pm(N),[pm(N)]).
solve(tp(N),[tp(N)]).
solve(tm(N),[tm(N)]).

solve(p(N),[pp(N1),X,pm(N1)]):-N1 is N-1, solve
(p(N1,N),X), !.
solve(m(N),[pp(N1),X,pm(N1)]):-N1 is N-1, solve
(m(N1,N),X), !.

solve(p(N,N1),[X,tm(N1),Y]):-N1 is N+1, N2 is N+2, solve
p(N1,N2),X), solve(m(N2),Y), !.
solve(m(N,N1),[Y,tp(N1),X]):-N1 is N+1, N2 is N+2,
solve(p(N2),Y), solve(m(N1,N2),X), !.

solve([],[]).
solve([X|Y], Z):-solve(X, X1), solve(Y, Y1), append (X1, Y1,
Z1), flatten(Z1, Z), !.

p_solve(X,Y):-solve(X,Y), write(Y).
```

図6　Proglogによるサンプルプログラム[20]

* 動作確認

先ず、プログラムコードを図5の問題用に書き換えて動作確認を行ってみよう。図6の最初の2文を
solve(p(2,3),[tm(3)]).
solve(m(2,3),[tp(3)]).
のように変えてエディタでプログラムを作成し、ディスクに保存する（デフォルトの拡張子は“pl”である[21]）。そして、プログラムファイルをダブルクリックするか、「File」+「Consult」コマンドでプログラムを起動して、コマンドライン（プロンプト“?-”の後：付録の「起動画面」参照）に「p_solve([m(3),m(2)],Y).」と入力すればよい（m(3)→m(2)は、解の仮想的な表現であることに注意）。画面には、
[pp(2), tp(3), pm(2), pp(1), pp(2), tm(3), pm(2), tp(2), tp(3), pm(1)]
のように表示され、(0)式と同様の結果となることが分かる。

6. 実行結果

五連環のプログラム（図6）の場合、仮想的な解は
pm(5)→m(4,5)→pm(4)→m(3,4)→pm(3)→m(2,3)→pm(2)→m(1,2)→pm(1)→p(2)→p(3)→p(4)→p(5)→tp(5)
と表現できる（(4)式）。

すなわち、先ず、上から下に向けて紐の右端を輪に通し、軸の上部を右から左に（仮想的に）通過させる。これを右上から左下にかけて繰り返すと、紐が軸［1］だけに懸かっている状態となる。次に、軸の下部を左から右にかけて（仮想的に）通過させ、最後に輪⑤の右横を通って大きく反時計回りに回して外せばよいわけである。

この仮想的な解に対応する下記のコマンドを入力すると具体的な解[22]が得られる（付録参照）。
p_solve([pm(5), m(4,5), pm(4), m(3,4), pm(3), m(2,3), pm(2), m(1,2), pm(1), p(2), p(3), p(4) ,p(5), tp(5)], Y).

得られた解を眺めてすぐに分かることは、無駄がとても多いということである。例えば、“pm(N), pp(N)”（紐を輪の中に通し元に戻す）というパターンが4箇所もある。そこで、その部分を削除すると、今度は“tp(N), tm(N)”（紐を輪の外に回り込ませ元に戻す）というパターンが4箇所生じ…といった具合に、いたるところに無駄な動きが組み込まれているようだ。

本来はこのような無駄を生じないアルゴリズムを作成した上でプログラミングし直すべきであるが、今回はエディタのマクロを使って安易な処理で済ませた[23]。以下がその結果である。
[pm(5), tp(5), pm(4), pm(3), tp(3), pp(4), tm(5), pm(4), tp(4), tp(5), pm(2), pm(1), pp(2), tm(5), tm(4), pp(4), tp(5), pm(4), tm(3), pp(3), pp(4), tm(5), pm(4), tp(4), tp(5), pm(3), pm(2),

[20] プログラム中、“append”, “flatten”, “write” の3つが組み込み述語である。

[21] Perlを使っている人には“迷惑な”拡張子名である。場合によっては何らかの措置が必要かも知れない。

[22] 「外れた紐を入れ直せてこそ知恵の輪が解けたことになる」という見解があるらしい。勿論、プログラムを再び動かすまでもなく、双対的な操作（解の矢印を逆向きにし、正負の向きを入れ替える）を求めれば済むことである。

[23] これでもPCを利用したことに変わりはない（単なるこじ付けの居直り！）。

pp(3), tm(5), tm(4), pp(4), tp(5), pm(4), pm(3), pp(4), tm(5), pm(4), tp(5)]

7. 2進法による解法

100円ショップで買った実際の五連環を使って確かめると確かに紐（実物は細長い針金）は外れる。しかし、実物で遊んでみると分かるが、「紐を動かす」というより「輪を（下向きに）外す」とか「輪を（上向きに）入れる」という方が実感に近い操作の表現である[24]。例えば、解の最初の部分“pm(5), tp(5)”は「輪⑤を外す」とみなせるし、その双対的な操作“tm(5), pp(5)”は「輪⑤を入れる」ことに相当する。

輪Ⓝを外すことをd(N)（“d”は“down”のつもり）と書き、輪Ⓝを入れることをu(N)（“u”は“up”のつもり）と書くことにし、輪の動きだけに注目してこの解を書き直すと、以下のようにさらに簡単になる。

[d(5), d(3), u(5), d(4), d(5), d(1), u(5), u(4), d(5), u(3), u(5), d(4), d(5), d(2), u(5), u(4), d(5), d(3), u(5), d(4), d(5)]

各輪について、輪が紐に“はまっている”状態を1、外れている状態を0と表現すると、五連環の“状態”を輪①～⑤の状態の並び（＝5桁の2進数）で表現できる。ただし、ここでは、輪の番号の大きいものほど下位のビットに対応すると考えることにする。

例えば、“11010”というのは、輪3と5だけが外れている状態である。実は、このような2進数を利用して、フランスのL.グロという人が1872年に九連環の問題を解いたということである[25]。

ここでは、グレイコード（Gray Code）による解法を紹介しよう[26]。

通常のバイナリコードと同じくグレイコードも0と1で数を表すものであるが、連続する2つの自然数のグレイコードは1つの桁しか異ならない[27]という性質を持つコードである。

バイナリコードからグレイコードへの変換は、列を右に1文字シフト[28]して、元の列とビットごとの排他的論理和（異なる数字なら1、等しい数字なら0）を取れば良い。例えば、“01101”の場合だと、右シフトした“00110”との排他的論理和“01011”が対応するグレイコードである。

グレイコードからバイナリコードへの変換は、以下の通りである： バイナリコードの最上位ビットはグレイコードと同じである。それ以外のバイナリコードのビットについては、ひとつ上位のバイナリコードのビットと同位のグレイコードのビットとの排他的論理和を取れば良い。

さて、五連環の場合、以下のような手順で解くことができる。ここで、“解く”というのは、変化する状態の推移を求めることを意味する。各状態はグレイコードとして示される。すなわち、初期状態は“11111”であり、目標とする最終状態は“00000”である。

1. グレイコード11111をバイナリコードに変換し、10101を得る（十進数で21）。
2. （十進）数列21, 20, …, 1, 0をグレイコードに変換したものが求める状態の推移である。

具体的には付録を参照のこと。

8. おわりに

少し調べてみただけだが、知恵の輪は奥が深いようだ。本文中でも触れたが、サンプルプログラムは不十分なものであって、無駄な操作を生じないアルゴリズムを考える必要がある。

また、本来の趣旨からいえば、（仮想的とはいえ）解をインプットするのは面白くないところである。もう少し人工知能的なものにしたければ、何らかのデータ構造で知恵の輪そのものの構造を記述し、それをインプットとして解（操作手順）を生成できるようにする必要がある。

例えば、購入した知恵の輪の中には、軸が何度か折れ曲がり、1つの輪の中に他の輪の軸が複数通っているものもあり、これまで調べてきた方法では全くのお手上げである（差分方程式を考えるより直接解く方がまだしも速いだろう）。

そして、紐と針金とは異なることにも注意したい。紐で実現できる動きが、物理的な制約のため、針金ではできないことが当然あり得る。ここで試みた解法が実物で可能だったことは、その意味では僥倖だったのだろう。

動きを紙面で説明するのはなかなか難しい。関心のある方は、実物[29]を使って是非体験していただき、もう少し“マシな”プログラムの作成にチャレンジしていただきたい。

---

[24] 以下は、文献[6]からの引用である（若干表現を変更した）。「1番右端の環は自由にはずせる。また2番目の環もはずすことができる。しかし、3番目から先は簡単にはいかない。一般にN番目の環は、(N－2)番目の環までは全部はずれていて、(N－1)番目の環がはまっているときに限ってはずすことができる。」
モデル（ここでは差分方程式）の側からこのような知見を得ることは難しい（後知恵的に証明することはできるのだろうが）。シンプルなモデルは美しいが、モデリングの過程で取りこぼした側面があることを意識しておくべきなのだろう。

[25] 文献[6]を参照。

[26] 文献[7]を参照。同文献には、操作の手間（最小の手数）についても説明されている。

[27] 2つのビット列の対応する桁のうち異なるビットの桁の数をハミング距離（Hamming distance）と呼ぶが、これはハミング距離が1ということである。

[28] シフト後の最上位ビットには0が入り、シフト前の最下位ビットは失われる。

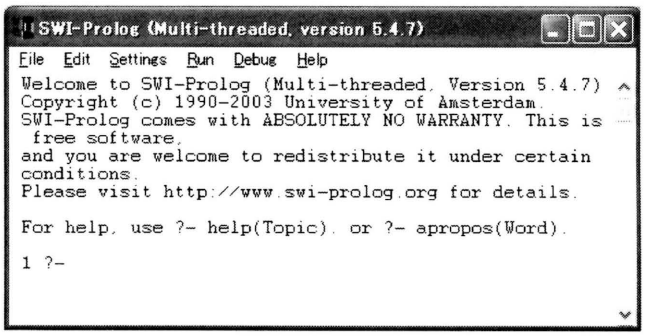
[29] ちなみに、使った五連環はダイソーの商品。

参考文献

[1] 石原秀男,「ループ＋再帰＝プログラム」, 専修ネットワーク＆インフォメーション, No.5, pp.1-8.
[2] ウィキペディア,「チャイニーズリング」の項 (http://ja.wikipedia.org/wiki/).
[3] カフェ・プロトマイヤ,「String Puzzle の数理的分析」(http://www.bas.nias.ac.jp/~cafe/hiroba/ hiroba.html),「パズルの解法」(JAVA applet) (http://www.bas.nias.ac.jp/~cafe/hiroba/math_a.html).
[4] 坂本實,「数理モデリング－概念・過程・教育」, 専修ネットワーク＆インフォメーション, No.2, pp.55-60.
[5] 杉本允生,「知的空間思考で遊ぶ知恵の輪の世界」, 新風社.
[6] 高木茂男,「パズル遊びへの招待 - オンライン版」, (http:// torito.jp/puzzles/104.html).
[7] 中村文則,「ハノイの塔で遊ぼう」 (http://www. nikonet.or.jp/spring/hanoi_n/hanoi_n2.pdf).
[8] 森克美,「解は存在します。しかし、その値は分かりません」, 専修ネットワーク＆インフォメーション, No.8, pp.95-99.
[9] John, R. Fisher, "Prolog:-Tutorial" (http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/contents.html#1).
[10] SWI-Prolog's Home, "What is SWI-Prolog?" (http://www.swi-prolog.org/).

付録

SWI-Prologの起動画面



五連環の解（プログラムによる出力）

[pm(5), tp(5), pm(4), pp(4), tm(5), pm(4), tp(4), tp(5), pm(3), pp(3), tm(5), tm(4), pp(4), tp(5), pm(4), pm(3), tp(3), pp(4), tm(5), pm(4), tp(4), tp(5), pm(2), pp(2), tm(5), tm(4), pp(4), tp(5), pm(4), tm(3), pp(3), pp(4), tm(5), pm(4), tp(4), tp(5), pm(3), pm(2), tp(2), pp(3), tm(5), tm(4), pp(4), tp(5), pm(4), pm(3), tp(3), pp(4), tm(5), pm(4), tp(4), tp(5), pm(1), pp(1), tm(5), tm(4), pp(4), tp(5), pm(4), tm(3), pp(3), pp(4), tm(5), pm(4), tp(4), tp(5), pm(3), tm(2), pp(2), pp(3), tm(5), tm(4), pp(4), tp(5), pm(4), pm(3), tp(3), pp(4), tm(5), pm(4), tp(4), tp(5), pm(2), pm(1), pp(2), tm(5), tm(4), pp(4), tp(5), pm(4), tm(3), pp(3), pp(4), tm(5), pm(4), tp(4), tp(5), pm(3), pm(2), pp(3), tm(5), tm(4), pp(4), tp(5), pm(4), pm(3), pp(4), tm(5), pm(4), tp(5)]

五連環の解（無駄を省いたもの）

[pm(5), tp(5), pm(4), pm(3), tp(3), pp(4), tm(5), pm(4), tp(4), tp(5), pm(2), pm(1), pp(2), tm(5), tm(4), pp(4), tp(5), pm(4), tm(3), pp(3), pp(4), tm(5), pm(4), tp(4), tp(5), pm(3), pm(2), pp(3), tm(5), tm(4), pp(4), tp(5), pm(4), pm(3), pp(4), tm(5), pm(4), tp(5)]

五連環の解（輪の操作として表現したもの）

[d(5), d(3), u(5), d(4), d(5), d(1), u(5), u(4), d(5), u(3), u(5), d(4), d(5), d(2), u(5), u(4), d(5), d(3), u(5), d(4), d(5)]

五連環の解（状態推移を表現したもの）

10進数.	状態	グレイコード	バイナリコード
21		11111	10101
20		11110	10100
19		11010	10011
18		11011	10010
17		11001	10001
16		11000	10000
15		01000	01111
14		01001	01110
13		01011	01101
12		01010	01100
11		01110	01011
10		01111	01010
9		01101	01001
8		01100	01000
7		00100	00111
6		00101	00110
5		00111	00101
4		00110	00100
3		00010	00011
2		00011	00010
1		00001	00001
0		00000	00000