5ゲーム、7ゲーム、8ゲームの研究

学習院大学理学部数学科4年

大久保 祐希

平成22年2月2日

目的

この研究では 15 ゲームを簡略化した 5 ゲーム、7 ゲーム、8 ゲームのやり方を定義し、それらを SWI-Prolog を用いた計算により置換群の立場から考察する。

まず 5 ゲーム、7 ゲーム、8 ゲームのやり方を定義する。15 ゲームにおいては縦横それぞれ 4×4 のマスとピースが与えられるが、5 ゲーム、7 ゲーム、8 ゲームにおいては、それぞれ 2×3、2×4、3×3 のマスとピースをそれぞれ与える。そこで次に 5 ゲームの場合について詳しく定義する。

準備として、空きスペースに 1、以下右上から横へ 2, 3,$\cdots$,6 という番号をマスとピース双方に与える。混乱のないように、マス（座席）の番号は $\underline{3}$ のように下線を引き、ピースの番号は [3] のように括弧をつけて区別する。[1] は、となりのピースと入れ換われる特別なピースを意味することとする（下の図 1 と次頁の図 2 を参照）

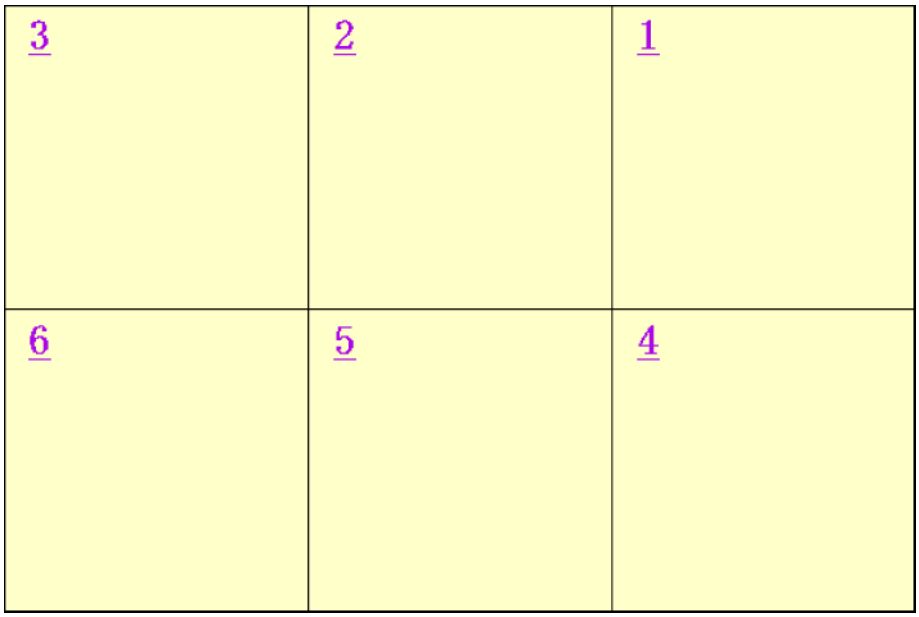

図 1: 各場所に右上から $\underline{1}$,$\underline{2}$,$\underline{3}$ $\cdots$, $\underline{6}$ と番号をつける

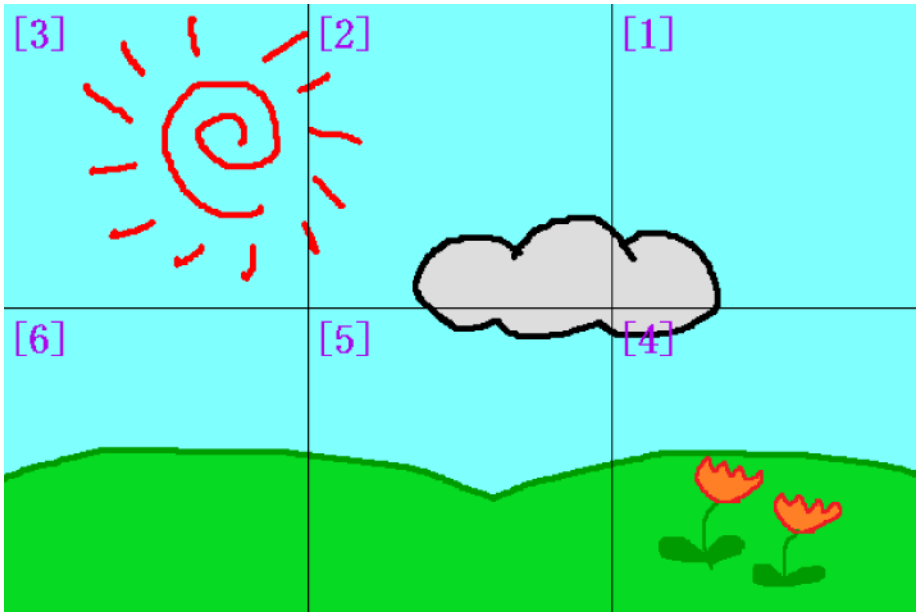

図 2: 各ピースに正しい絵柄の右上から [1],[2],[3],$\cdots$, [6] と番号をつける

(i) いま 6 文字の置換

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ i_1 & i_2 & i_3 & i_4 & i_5 & i_6 \end{pmatrix}$$

を、

座席 1 にいるピースを i_1 に移し、座席 2 にいるピースを i_2 に移す。以下同様の方法で 6 個のピースの座席を換える

という意味に解釈することと定める。

(ii) ゲームを進行させる操作は、1 回ずつ、[1] とその隣にいるピースとの入れ換えを続けることとする。したがって、この 1 回のアクションは互換である。たとえば場所 $\underline{1}$ に [1] がある場合の次の置換

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 2 & 3 & 1 & 5 & 6 \end{pmatrix}$$

は $\underline{1}$ にある [1] と $\underline{4}$ にあるピースを入れ替えるアクションである（次頁の図 3 を参照）

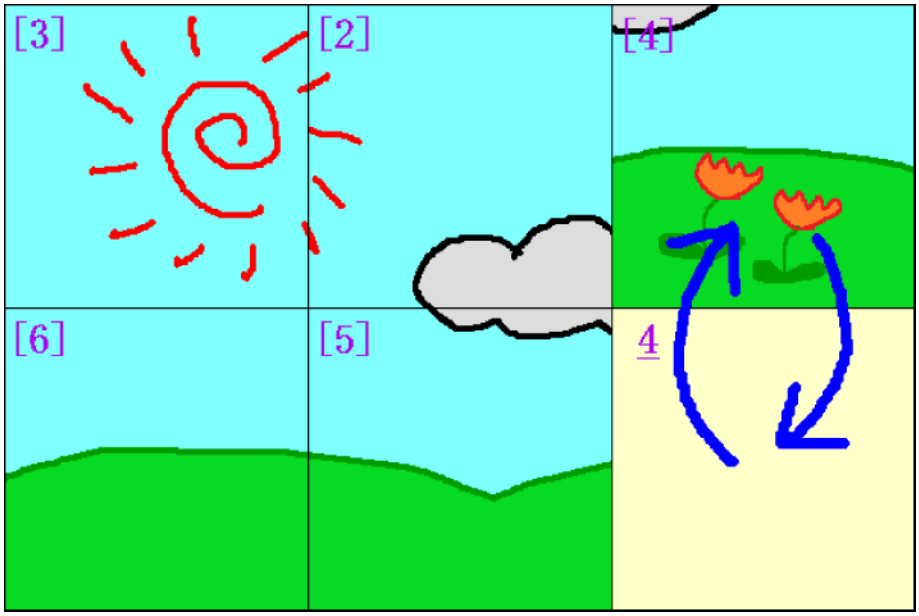

図 3: [1] マスとその隣のマスを入れ替えるアクション 1 回は互換である

(iii) 最初の配置から最終の配置への変換は、ある 6 文字の置換

$$\sigma_5 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ i_1 & i_2 & i_3 & i_4 & i_5 & i_6 \end{pmatrix}$$

と表すものとする。そのとき最終形の場所 $\underline{1}$ には常に空きスペース [1] がくるものとする。すなわち、$\sigma(1) = 1$ である。

このゲームで生じる 6 文字の置換 σ_5 を 5 ゲーム置換と呼ぶこととする。

(iv) 下の図 4 のように各マスを市松模様に塗り分けておくと、1 回のアクションで空ピース [1] は必ず白マスから黒マスへ、あるいは黒マスから白マスへ移る。したがって空ピース [1] は最初右上の白マスにあり、最終形でも右上の白マスにいるわけだから（$\sigma(1)=1$ より）最終形にいたるアクションの回数は必ず偶数である。すなわち、

5 ゲーム置換 σ_5 は偶置換である.

縦横が 2×4 の 7 ゲーム、3×3 の 8 ゲームにおいても白黒の市松模様を塗り分けると 5 ゲームと同様に偶置換であると確認できる。

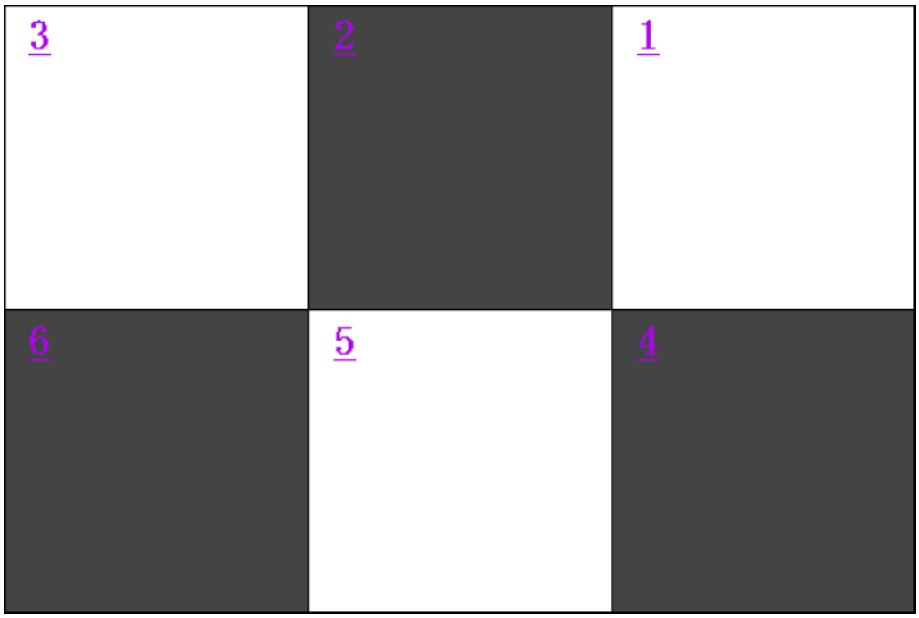

図 4: 最初右上の白地にある [1] は最終形でも右上の白地に移動する。アクション回数は偶数である

(v)5 ゲーム置換 σ_5 において、ここで (ii) で説明した互換アクションを繰り返して $\underline{1}$ にある [1] を 2×3 のマス全体で時計回りに 1 周させ、再び $\underline{1}$ に戻るまで動かして絵柄全体をかき混ぜる操作をする。[1] と入れ替わる各ピースの流れは反時計回りとなる。ここでその置換 τ_{5_1} を

$$\tau_{5_1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 3 & 6 & 2 & 4 & 5 \end{pmatrix}$$

と定める。ここでは仮に『5 ゲームの外回り置換』と呼ぶこととする。また $\underline{1}$ にある [1] を、$\underline{3}$ と $\underline{6}$ は通らず、 $\underline{4}, \underline{5}, \underline{2}$ の内回りで [1] を時計回りに 1 周させる置換 τ_{5_2} を

$$\tau_{5_2} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 5 & 3 & 2 & 4 & 6 \end{pmatrix}$$

と定める。ここでは仮に『5 ゲームの内回り置換』と呼ぶこととする。この τ_{5_1}、τ_{5_2} の操作を複数回行うことによって、この 5 ゲームの操作で生じる全ての絵柄のパターンを生成できるかを次の『方法』の項で計算し確認する。(下の図 5 と次頁の図 6 を参照)

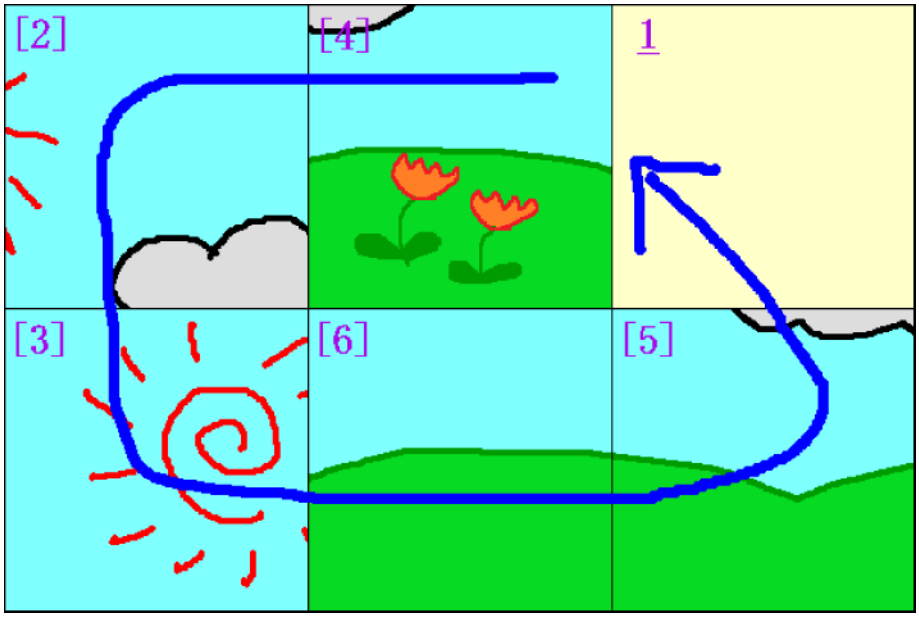

図 5: 5 ゲームの外回り置換

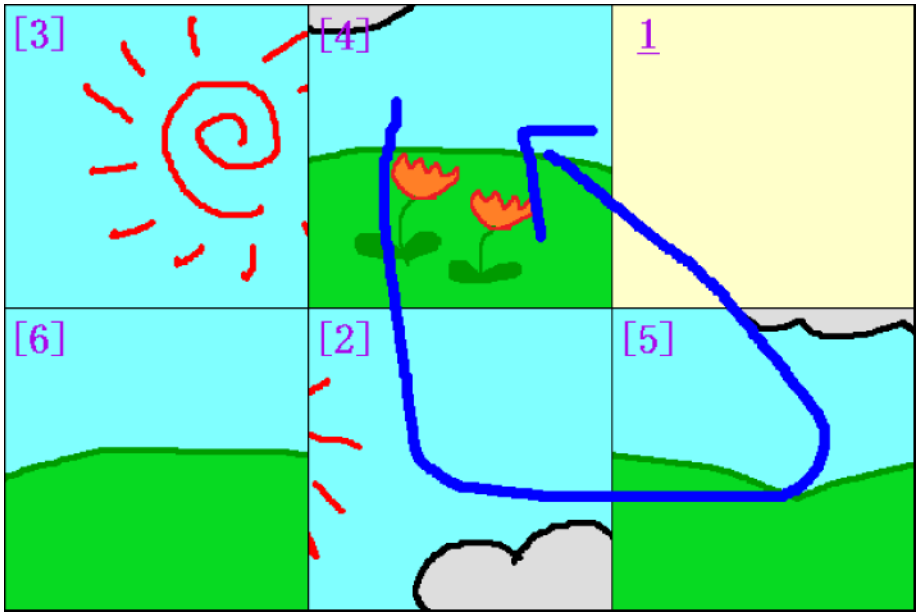

図 6: 5 ゲーム内回り置換

(vi)5 ゲームと同様に 7 ゲーム置換 σ_7 と 8 ゲームの置換 σ_8 を以下のように定める。

$$\sigma_7 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ i_1 & i_2 & i_3 & i_4 & i_5 & i_6 & i_7 & i_8 \end{pmatrix}$$

$$\sigma_8 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ i_1 & i_2 & i_3 & i_4 & i_5 & i_6 & i_7 & i_8 & i_9 \end{pmatrix}$$

7 ゲームの外回りと内回りの置換 τ_{7_1}、τ_{7_2}、8 ゲームの外回りと内回りの置換 τ_{8_1}、τ_{8_2}、を以下のように定める（下の図 7 と次頁の図 8 を参照）

$$\tau_{7_1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 4 & 8 & 2 & 5 & 6 & 7 \end{pmatrix}$$

$$\tau_{7_2} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 6 & 3 & 4 & 2 & 5 & 7 & 8 \end{pmatrix}$$

$$\tau_{8_1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 3 & 6 & 2 & 5 & 9 & 4 & 7 & 8 \end{pmatrix}$$

$$\tau_{8_2} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 5 & 3 & 2 & 4 & 6 & 7 & 8 & 9 \end{pmatrix}$$

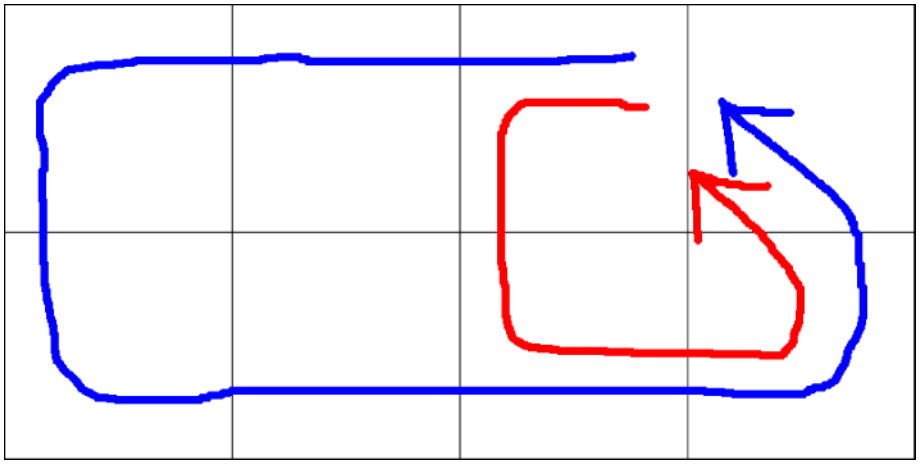

図 7: 7 ゲームの外回りと内回りの置換

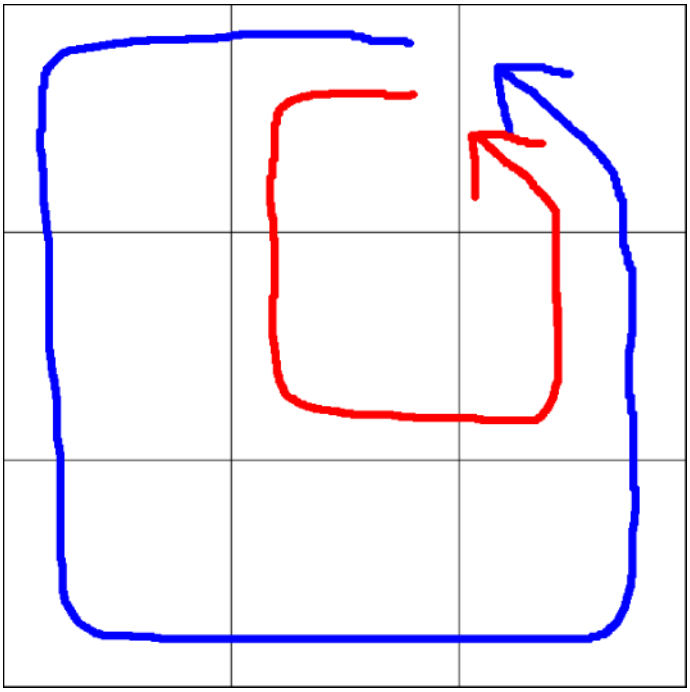

図 8: 8 ゲームの外回りと内回りの置換

7 ゲーム、8 ゲームにおいても上記の外回りと内回りの操作を複数回繰り返すことによって生じうる全ての絵柄のパターンを生成できるかを次の『方法』の項で計算し確認する。

方法

先の『目的』の項目で述べた定義をもとにSWI-Prologを用いて計算する。以下にそのプログラムを記述する。

```
------------------------------------------------------------

copy(X=X).

not(Q):- Q,!,fail.
not(_).

round_out5(A):-  copy( A=[1,3,6,2,4,5] ).
round_in5(A):-   copy( A=[1,5,3,2,4,6] ).

round_out7(A):-  copy( A=[1,3,4,8,2,5,6,7] ).
round_in7(A):-   copy( A=[1,6,3,4,2,5,7,8] ).

round_out8(A):-  copy( A=[1,3,6,2,5,9,4,7,8] ).
round_in8(A):-   copy( A=[1,5,3,2,4,6,7,8,9] ).

r_out5(A):- copy( A=[10] ).
r_in5(A):-  copy( A=[11] ).

subs_product([]=_*[]).
subs_product([B|List]=Tau1*[A|Tau2]):-
    subs_pro_aux(B=Tau1*A,1),
    subs_product(List=Tau1*Tau2).

subs_pro_aux(_=[]*_,_):-fail.
subs_pro_aux(B=[B|_]*A,A):- !.
subs_pro_aux(Res=[_|Tau1]*A,S):- S1 is S+1,
    subs_pro_aux(Res=Tau1*A,S1).

:-dynamic s/1.
:-dynamic s2/2.
:-dynamic h2/2.

listup(A,B):- s(A),s(B).
listup(_,_):- fail.

list_head(Z=[Z|_]).
```

```
list_tail(Z=[Z]).
list_tail(Z=[_|List]):-list_tail(Z=List).

listup_s(A1,AS,B1,BS):- s2(A1,AS),s2(B1,BS).
listup_s(_,_,_,_):- fail.

listup_h(A2,AS,B2,BS):- h2(A2,AS),h2(B2,BS),!.
listup_h(_,_,_,_):- fail.

game5 :-
    abolish(s2/2),
    abolish(h2/2),

    round_out5(X1),
    round_in5(Y1),

    r_out5(X2),
    r_in5(Y2),

    assertz(s2(X1,1)),
    assertz(s2(Y1,2)),

    assertz(h2(X2,1)),
    assertz(h2(Y2,2)),

    write(1),put(9),write(X1),put(9),youso_write(X2,0),nl,
    write(2),put(9),write(Y1),put(9),youso_write(Y2,0),nl,

    game5_aux(3),fail.
game5.

game5_aux(S) :-
    listup_s(A1,AS,B1,BS),
    listup_h(A2,AS,B2,BS),

    subs_product(C1=A1*B1),
    youso5(C2=A2+B2),

    not(s2(C1,_)),
    assertz(s2(C1,S)),assertz(h2(C2,S)),!,
    write(S),put(9),write(C1),put(9),youso_write(C2,0),nl,
```

```
    S1 is S+1,
    game5_aux(S1).
game5_aux(_).

youso_sum5(B=C+D):- (C mod 10 =:= 0),B is (C+D-(C mod 10)) mod 50.
youso_sum5(B=C+D):- (C mod 10 =:= 1),B is (C+D-(C mod 10)) mod 30.

youso5([A|List]=[A|L]+L2):-
    youso5(List=L+L2).
youso5([B|L]=[C]+[D|L]):- (C mod 10 =:= D mod 10),youso_sum5(B=C+D).
youso5([E|[F|L]]=[E]+[F|L]):- (E mod 10 =\= F mod 10).

if_write(A):- (A mod 10) =:= 0, write('X^'),
    B is ((A -(A mod 10))/10),write(B).
if_write(A):- (A mod 10) =:= 1, write('Y^'),
    B is ((A -(A mod 10))/10),write(B).
if_write(_):- fail.

youso_write([A|List],0):- if_write(A),youso_write(List,1).
youso_write([A|List],1):- write(' * '),if_write(A),youso_write(List,1).
youso_write([],_).

game7 :-
    abolish(s/1),

    round_out7(X),
    round_in7(Y),

    asserta(s(X)),
    asserta(s(Y)),

    write(1),nl,
    write(2),nl,

    game7_aux(3),fail.
game7.

game7_aux(S) :-
    listup(A,B),
    subs_product(C=A*B),

    not(s(C)),asserta(s(C)),!,
    write(S),nl,
```

```
    S1 is S+1,
    game7_aux(S1).
game7_aux(_).

game8 :-
    abolish(s/1),

    round_out8(X),
    round_in8(Y),

    asserta(s(X)),
    asserta(s(Y)),

    write(1),nl,
    write(2),nl,

    game8_aux(3),fail.
game8.

game8_aux(S) :-
    listup(A,B),
    subs_product(C=A*B),

    not(s(C)),asserta(s(C)),!,
    write(S),nl,

    S1 is S+1,
    game8_aux(S1).
game8_aux(_).
```

プログラム内において、例えば置換

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 4 & 6 & 2 & 5 & 3 \end{pmatrix}$$

はリストを用いて単純に

```
[1,4,6,2,5,3]
```

と表現する。subs_product 述語は引数として同じ長さの 2 つの置換を右辺に挿入し、その積を返す。game5 述語によってまず外回りの置換 [1,3,6,2,4,5]、内回りの置換 [1,5,3,2,4,6] をそれぞれ初期解としてプログラムデータベースに書き込む。以下、プログラムデータベースに書き込まれた解から 2 つを任意に選び、それらの積を新たな解としてプログラムデータベースに書き込む（既に書き込まれている場合は失敗し、別の 2 つの初期解から新たな解を探す）これを解が出尽くすまで続ける。

また game5 述語においては外回り置換を X、内回り置換を Y とおき、それぞれの解が外回りと内回りをそれぞれ何回掛け合わせることによって生成されたかを可視化できるようにした。以上をもとに game5 述語を質問して結果を載せる。

game5 述語を質問する。

```
1 ?- game5.

1       [1, 3, 6, 2, 4, 5]      X^1
2       [1, 5, 3, 2, 4, 6]      Y^1
3       [1, 6, 5, 3, 2, 4]      X^2
4       [1, 4, 6, 3, 2, 5]      X^1 * Y^1
5       [1, 5, 4, 6, 3, 2]      X^3
6       [1, 2, 5, 6, 3, 4]      X^2 * Y^1
7       [1, 4, 2, 5, 6, 3]      X^4
8       [1, 3, 4, 5, 6, 2]      X^3 * Y^1
9       [1, 2, 3, 4, 5, 6]      X^0
10      [1, 6, 2, 4, 5, 3]      X^4 * Y^1
11      [1, 3, 6, 5, 2, 4]      Y^1 * X^1
12      [1, 6, 5, 4, 3, 2]      X^1 * Y^1 * X^1
13      [1, 5, 4, 2, 6, 3]      X^2 * Y^1 * X^1
14      [1, 4, 2, 3, 5, 6]      X^3 * Y^1 * X^1
15      [1, 2, 3, 6, 4, 5]      X^4 * Y^1 * X^1
16      [1, 4, 3, 5, 2, 6]      Y^2
17      [1, 2, 6, 4, 3, 5]      X^1 * Y^2
18      [1, 3, 5, 2, 6, 4]      X^2 * Y^2
19      [1, 6, 4, 3, 5, 2]      X^3 * Y^2
20      [1, 5, 2, 6, 4, 3]      X^4 * Y^2
21      [1, 2, 6, 3, 5, 4]      Y^1 * X^1 * Y^1
22      [1, 3, 5, 6, 4, 2]      X^1 * Y^1 * X^1 * Y^1
23      [1, 6, 4, 5, 2, 3]      X^2 * Y^1 * X^1 * Y^1
24      [1, 5, 2, 4, 3, 6]      X^3 * Y^1 * X^1 * Y^1
25      [1, 4, 3, 2, 6, 5]      X^4 * Y^1 * X^1 * Y^1
26      [1, 4, 2, 6, 3, 5]      Y^1 * X^3
27      [1, 2, 3, 5, 6, 4]      X^1 * Y^1 * X^3
```

```
28      [1, 3, 6, 4, 5, 2]      X^2 * Y^1 * X^3
29      [1, 6, 5, 2, 4, 3]      X^3 * Y^1 * X^3
30      [1, 5, 4, 3, 2, 6]      X^4 * Y^1 * X^3
31      [1, 2, 5, 4, 6, 3]      Y^1 * X^4
32      [1, 3, 4, 2, 5, 6]      X^1 * Y^1 * X^4
33      [1, 6, 2, 3, 4, 5]      X^2 * Y^1 * X^4
34      [1, 5, 3, 6, 2, 4]      X^3 * Y^1 * X^4
35      [1, 4, 6, 5, 3, 2]      X^4 * Y^1 * X^4
36      [1, 3, 2, 4, 6, 5]      Y^1 * X^3 * Y^1
37      [1, 6, 3, 2, 5, 4]      X^1 * Y^1 * X^3 * Y^1
38      [1, 5, 6, 3, 4, 2]      X^2 * Y^1 * X^3 * Y^1
39      [1, 4, 5, 6, 2, 3]      X^3 * Y^1 * X^3 * Y^1
40      [1, 2, 4, 5, 3, 6]      X^4 * Y^1 * X^3 * Y^1
41      [1, 6, 4, 2, 3, 5]      Y^1 * X^1 * Y^1 * X^1
42      [1, 5, 2, 3, 6, 4]      X^1 * Y^1 * X^1 * Y^1 * X^1
43      [1, 4, 3, 6, 5, 2]      X^2 * Y^1 * X^1 * Y^1 * X^1
44      [1, 2, 6, 5, 4, 3]      X^3 * Y^1 * X^1 * Y^1 * X^1
45      [1, 3, 5, 4, 2, 6]      X^4 * Y^1 * X^1 * Y^1 * X^1
46      [1, 2, 5, 3, 4, 6]      Y^1 * X^3 * Y^1 * X^1
47      [1, 3, 4, 6, 2, 5]      X^1 * Y^1 * X^3 * Y^1 * X^1
48      [1, 6, 2, 5, 3, 4]      X^2 * Y^1 * X^3 * Y^1 * X^1
49      [1, 5, 3, 4, 6, 2]      X^3 * Y^1 * X^3 * Y^1 * X^1
50      [1, 4, 6, 2, 5, 3]      X^4 * Y^1 * X^3 * Y^1 * X^1
51      [1, 5, 6, 2, 3, 4]      Y^1 * X^1 * Y^2
52      [1, 4, 5, 3, 6, 2]      X^1 * Y^1 * X^1 * Y^2
53      [1, 2, 4, 6, 5, 3]      X^2 * Y^1 * X^1 * Y^2
54      [1, 3, 2, 5, 4, 6]      X^3 * Y^1 * X^1 * Y^2
55      [1, 6, 3, 4, 2, 5]      X^4 * Y^1 * X^1 * Y^2
56      [1, 4, 5, 2, 3, 6]      Y^1 * X^3 * Y^1 * X^1 * Y^1
57      [1, 2, 4, 3, 6, 5]      X^1 * Y^1 * X^3 * Y^1 * X^1 * Y^1
58      [1, 3, 2, 6, 5, 4]      X^2 * Y^1 * X^3 * Y^1 * X^1 * Y^1
59      [1, 6, 3, 5, 4, 2]      X^3 * Y^1 * X^3 * Y^1 * X^1 * Y^1
60      [1, 5, 6, 4, 2, 3]      X^4 * Y^1 * X^3 * Y^1 * X^1 * Y^1
true.
```

確認してみるとわかるが出力された 60 の解のうち全く同じ置換は 1 つも存在しない。5 ゲームの全てのパターンの数は $\sigma(1) = 1$ の部分を除いた 5 つの部分の組み合わせから奇置換の数を除いたものである。その数は $5!/2 = 60$ となり、プログラムの計算によって出た結果と一致する。よってこのプログラムにおける 5 ゲームの計算は成功したものと見なすことができる。

続けて 7 ゲームと 8 ゲームに関しても計算する。ただし 7 ゲームと 8 ゲームは、5 ゲームと比べて計算量が多いためリストによる置換の表示はせず解の数のみ表示するものとする。

game7 述語を質問する。

```
2 ?- game7.

1
2
3
.
.
.
(中略)
.
.
.
2518
2519
2520
```

本来なら最後の行に true. が出てプログラムが止まるはずなのだが、Prolog の予期せぬ動作でここで止まってしまう。しかし 7 ゲームの予想しうる全てのパターンの数、$7!/2 = 2520$ に一致した。

game8 述語を質問する。

```
2 ?- game8.

1
2
3
.
.
.
(中略)
.
.
.
20158
20159
20160
```

こちらも true. が出ずに止まってしまうが、8 ゲームの予想しうる全てのパターンの数、$8!/2 = 20160$ に一致した。8 ゲームの外回りの置換では中心のマス（5 のマス）が動かないが、内回りの置換で中心のマスを動かすことによって全てのマスをカバーし、全てのパターンを算出できた点は特筆すべきだろう。

また 7,8 ゲームにおいて、試みに内回りの置換を変えて計算したらどうなるかを調べた。7 ゲームにおいて τ'_{7_1}, τ'_{7_2} を以下のように定め計算してみたところ、全ての絵柄パターンが生成された。

$$\tau'_{7_1} = \tau_{7_1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 4 & 8 & 2 & 5 & 6 & 7 \end{pmatrix}$$

$$\tau'_{7_2} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 7 & 4 & 2 & 5 & 6 & 8 \end{pmatrix}$$

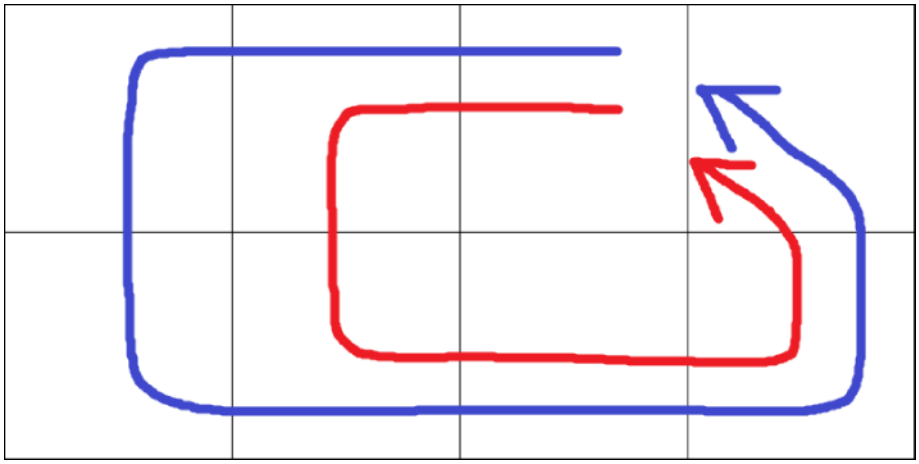

図 9: τ'_{7_1}, τ'_{7_2} をこのようにしても全ての絵柄パターンは生成される

8 ゲームにおいて τ'_{8_1}, τ'_{8_2} を以下のように定め計算してみたところ、全ての絵柄パターンが生成された。

$$\tau'_{8_1} = \tau_{8_1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 3 & 6 & 2 & 5 & 9 & 4 & 7 & 8 \end{pmatrix}$$

$$\tau'_{8_2} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 3 & 6 & 2 & 4 & 5 & 7 & 8 & 9 \end{pmatrix}$$

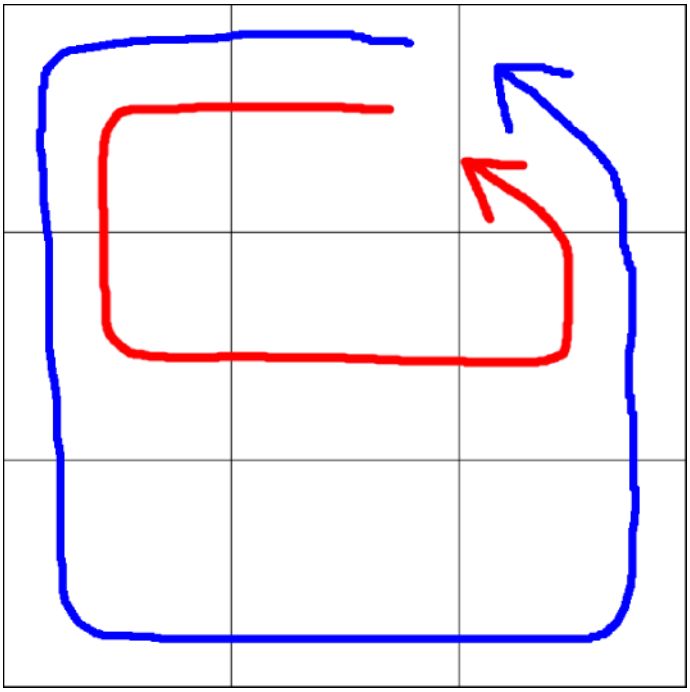

図 10: τ'_{8_1}, τ'_{8_2} をこのようにしても全ての絵柄パターンは生成される

8 ゲームにおいて上記の例以外にも全ての絵柄パターンを再現できる外回りと内回りの組み合わせが存在するが、ここでは省略する。

考察

5 ゲームにおいて、5 ゲーム置換群を A_5 とおくと

$$A_5 = \langle (234), (235), (236) \rangle$$

が成り立つ。τ_{5_1} と τ_{5_2} はこれら 3 サイクルの積により生成されており

$$\tau_{5_2} = (254) = (235)^2 (234)^2 (235)$$

$$\tau_{5_1} = (23654) = (235)^2 (234)^2 (235) (236)$$

が成り立つ。$(234), (235), (236)$ から生成される τ_{5_1},τ_{5_2} から A_5 が確かに生成された、ということをプログラムが結果をもって示したこととなる。これは 7,8 ゲームそれぞれにおいても同様のことがいえる。

前頁より τ_{7_1}, τ_{7_2} と τ_{8_1},τ_{8_2} にはそれぞれ複数の組み合わせがあるが、上記の 3 サイクルの積の掛け合わせの仕方が重要なのではなく、何個の 3 サイクルの積から生成されるかが重要であることを確認できる。

今後の課題

今回の研究ではゲームにおける全ての絵柄パターンを算出することに終始しましたが、バラバラの絵柄から最短で元の絵柄に戻す問題にも発展できそうだと思います。後続で研究する人がいるとするなら是非とも頑張ってほしいと思います。

感想

現状のプログラムでは game7 述語と game8 述語を動かすと解は出るもののプログラムが止まってしまい、また game5 述語においても X と Y の組み合わせは出たもののその組み合わせ方の設定などは出来ず有用な法則を発見できませんでした。プログラムの組み方において反省点があったと思います。

今回の研究を通じて交代群の定理が実際の場面でどういう形で現れるかを知ることができ、数学がより身近なものに感じられ非常に意義があったと思います。飯高先生には SWI-Prolog のプログラミング、数学的な内容ともに大変お世話になりました。1 年間本当にありがとうございました。