GAP の遊び方

庄 司 俊 明

名古屋大学大学院多元数理科学研究科

目　　次

GAP の遊び方

GAP (Group, Algorithm and Programming) は, 環, 体などの抽象代数の計算のための専用ソフトウエアです. 無料で公開されています. ここでは GAP4 のバージョンを使います. GAP により有限群や, 有限体, 有理数体, 代数体, あるいはその上のベクトル空間における種々の計算が可能です. あくまで整数計算が基本ですから実数体や複素数体での数値計算には向きません.

GAP を個人的に取り寄せたい場合は, 以下のサイトから ダウンロードできます.

```
http://www-gap.dcs.st-and.ac.uk/gap
```

とそのミラーサイト

```
http://www.math.rwth-aachen.de/~GAP
```

```
http://www.ccs.neu.edu/mirrors/GAP
```

```
http://wwwmaths.anu.edu.au/research.groups/algebra/GAP/www/
```

また, GAP のマニュアルは次のサイトにあります.

```
http://www.gap-system.org/Manuals/doc/htm/ref/
```

1 章　GAP World への招待

1.1 GAP の始め方と終り方

GAP をスタートさせるには、ターミナル上で `gap` とタイプすれば始まる.

```
% gap
```

ここで return key を押すと, GAP のロゴと共に

```
gap>
```

が画面に表れて, 入力待ちの状態になる.

GAP を終らせるためには, `quit;` とタイプする. 重要なことは, 命令を書いたら必ず最後にセミコロン ; をつけることである. GAP では return key は命令の終了ではなく, 単に命令待ちを意味する.

```
gap> quit;
%
```

となり, GAP が終了する. 機種によっては, `quit;` の代わりに *ctl*-D (コントロールキーを押しながら D を押す) によっても終了できる. 試してみて下さい. また, 何らかの事情で GAP の進行を停止させたい時は *ctl*-C を連続して 2 回打てば良い.

1.2 GAP による計算記録の残し方

GAP は対話型のソフトウエアなので, 画面上でコンピュータと交信しながら計算を進めて行く. その計算結果を残しておかないと, GAP を終了したときに計算結果が失われてしまう. 簡単に記録を残す方法は以下のようにログファイルに結果を書き込むことである.

```
gap> SizeScreen( [ 72, ] ); LogTo( "sample.log" );
```

これは, セミコロン ; で区切られたふたつの命令を表す. 最初の命令は, 計算結果を一行 72 のサイズで画面に打ち出すこと, 次の命令は, 画面に表れる文字や数字をそのままの形で `sample.log` というファイルに書き出すこと, である. 二つの命令は全く独立に書いても構わない. 計算に対する指令から計算結果まで, `quit;` により GAP が終るまで画面が全て `sample.log` に保存される. GAP を終えた後、このファイルを参照すれば, 計算結果が復元される. このファイルは編集可能なので, 後にファイルの不必要な部分を消し去れば有効にデータが保存される. なお, GAP を何回か動かしそれを同じ名前のログファイルに保存していけば新しいデータがファイルにどんどん追加されて行くことになる.

1.3 GAP による簡単な計算

まず整数の足し算, 引き算, かけ算をやってみよう. 例えば, $(9-7)\times(5+6)$ は `(9 - 7) * (5 + 6)` とタイプする. GAP は以下のように計算結果を返す.

```
gap> (9 - 7) * (5 + 6);
22
gap>
```

ここでもし, 最後のセミコロンを忘れて return key を押すと GAP は `>` のみを出力し沈黙を守る. GAP は更なる入力をひたすら待ち続けるのである. そこで, `>` の後に, セミコロン; を入力すると,

```
gap> (9 - 7) * (5 + 6)
>;
22
```

と以前の計算が復活する. しかし, 例えば最後のカッコ) を忘れたりすると, これは少し罪が重い. GAP は「ご主人様ごめんなさい, できません」 というわけで,

```
gap> (9 - 7)*(5 + 6;
Syntax error: ) expected
(9 - 7)*(5 + 6;
              ^
```

と, 間違いの箇所を推測し, その位置を指摘してくれる. Syntax error とは文法上の間違いである.

時に, 文法上のエラーから無限に続く迷路に入ってしまうことがある. GAP は, 決して見つかることのない答を求めて闇をさすらい, 無益な計算をし続けるのである. これを**ブレイクループ** (break loop) (日本語では, 「悪循環」??) といい, `brk>` という表示が出る. ブレイクループの中で, さらにエラーが起これば, `brk_02>`, `brk_03>` というように事態はどんどん悪化していく. ブレイクループから抜け出すには,

```
brk> quit;
```

とすればよい. あるいは *ctl*-D としてもよい. GAP のプロンプト `gap>` が復活する.

次に割算について見てみよう. 前にも書いたように, GAP は小数の計算には興味を示さない. あくまで, 分数の世界にこだわるのである. GAP は, 与えられた分数を約分し, 既約分数で表す. さらに, 頭の良いことには, 分数の足し算をやらせれば、通分して答えを既約分数の形で返してくる.

```
gap> 12345/25;
2469/5
gap> 123/4567 + 234/5678;
883536/12965713
```

巾乗の計算, 例えば 3^{132} は `3^132` とタイプする.

```
gap> 3^132;
955004950796825236893190701774414011919935138974343129836853841
```

整数の割算による余りは `mod` で計算できる. 17 を 3 で割ると余り 2 となるので

```
gap> 17 mod 3;
2
```

が得られる.

また, GAP は命題や数式が正しいか (`true`), 間違っているか (`false`) を判定する. これを論理演算という.

```
gap> (9 - 7)*5 = 9 - 7*5;
false
```

数の大小の比較には, `=`, `<>`, `<`, `<=`, `>`, `>=` が使われる. それぞれ, $a = b, a \neq b$, $a < b, a \leq b, a > b, a \geq b$ の意味である. 論理演算とあわせて,

```
gap> 10^5 < 10^4;
false
```

より複雑な論理演算は次のようになる.

```
gap> not true; true and false; true or false;
false
false
true
```

それぞれ, 「正しい」の反対は間違い, 「正しくて、同時に間違い」ということはない, 「正しいかまたは間違っている」というのは常に正しい, を意味する. それでは, 次はどうなるか. 答えを予測してからやってみよう.

```
gap> not true or false and not not false or true and false;
```

GAP では, 文字を定数としてそのまま扱うことができる. 以下に述べる文字変数とは区別する必要がある. 文字 a や $*$ を定数として使いたい場合は'a' あるいは '*' と両側から' ではさむ.

```
gap> 'a';
'a'
gap> '*';
'*'
```

ただし、'ab' とはできず, 一文字のみ. GAP は文字定数が等しいか異なるかを判定する.

1.4 群演算の簡単な例

対称群 S_n の元 σ は整数 $I_n = \{1, 2, \ldots, n\}$ の置換

$$\sigma = \begin{pmatrix} 1 & 2 & \cdots & n \\ \sigma(1) & \sigma(2) & \cdots & \sigma(n) \end{pmatrix}$$

として表される. $\sigma \in S_n$ は共通な文字を含まない巡回置換の積として一意的に表される. 例えば,

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 5 & 1 & 3 \end{pmatrix} = (124)(35).$$

ここで例えば (124) は $1 \to 2 \to 4 \to 1$ となる巡回置換を表す.

GAP では, 対称群の計算がそのままの記号で計算できる. (1,2,3) と入力すると, GAP はこれを S_3 の巡回置換 $\sigma = (123)$ として認識する. σ の逆元 σ^{-1} は巡回置換 (132) になるが, GAP ではこれを

```
gap> (1,2,3)^-1;
(1,3,2)
```

と計算してくれる. 見れば分かるように GAP では`^-1` を -1 乗, すなわち逆元として素直に理解してくれる (普通なかなかそうは行かない).

文字 2 に $\sigma = (123)$ が作用して, $\sigma(2) = 3$ となるのは, GAP では

```
gap> 2^(1,2,3);
3
```

と計算される. $\sigma(2) = 3$ の代わりに $2^{\sigma} = 3$ と書くことに注意する. 本来, ${}^{\sigma}2 = 3$ と書かないと左作用にならないが, 後者の書き方はタイプしにくいので, 前者を採用したい. そのために, S_n の I_n への作用は右作用を考える. この場合, S_n の積の計算は左から右へとすることになる. たとえば

```
gap> (1,2)*(2,3);
(1,3,2)
```

となる. 左作用で計算すれば $(12)(23) = (123)$ となるはずである.

また, 数の列 $(1, 2, 3)$ に $\sigma = (1, 2)$ を置換として作用させると,
$(1, 2, 3)\sigma = (\sigma(1), \sigma(2), \sigma(3)) = (2, 1, 3)$ となるが, GAP では

```
gap> (1,2,3)^(1,2);
(2,1,3)
```

となる.

1.5 変数とオブジェクト

GAP では**変数**を定義し, その変数に何か (数とは限らない) ある物を**代入** することができる. 一般に, 変数に代入される物のことを **オブジェクト** という. 例えば, 文字 a を変数として定義する場合, `a:=` と書く. (`a :=` でもよい. ただし `: =` の様に, 間にスペースを入れてはいけない).

```
gap> a:= (9 - 7)*(5 + 6);
22
gap> a;
22
gap> a*(a+1);
506
gap> a = 10;
false
gap> a:= 10;
10
gap> a*(a+1);
110
```

最初の式で, 変数 a が定義され, そこに $(9-7)\times(5+6)=22$ が値として代入される. 2 番目の式のように, 変数 a を呼び出せば, GAP はその値 22 を返す. 3 番目の式では, $a=22$ として $a(a+1)$ を計算し, その値 506 が得られる. また a に新しい値 10 を代入するには, a:= 10 と書けばよい. ここで a = 10 では代入にならないことに注意する. GAP では = は等式が正しいか間違っているかの判定に使われる. 4 番目の場合, a には値 22 が入っているので, $a=10$ は誤りであり, GAP は false と答える.
GAP では, 値を代入すると常に次の行にその値を書く.

```
gap> w:= 2;
2
```

いちいち変数の値を書く必要がない場合（例えばプログラムを作る場合など), 次のようにセミコロンを２つ続けて書くと, 変数の値が次の行に書かれない.

```
gap> w:= 2;;
gap> w*(w+2);
8
```

GAP では, 基本的にまぎらわしくない限り, どんな文字や数字も変数として使える. たとえば abc や a0bc1. また, GAP は大文字と小文字を区別するので, a1 と A1 は異なる変数を表わす. しかし 1234 は数字 1234 と区別がつかないので, 変数としては使えない.

また 例えば quit は, 終了のコマンドとして登録されているので変数としては使えない. このような言葉は “キーワード” として区別される. さらに, GAP はそのライブラリに多くの変数や, 関数を持っている. これらの言葉は変数として使えない. しかし, GAP が定義している関数はすべて大文字で始まる (次節参照. 例えば, Factorial, Gcd, Print など). そこで小文字で始まる変数を定義する限りは, GAP の変数や関数と重複する心配はない.

1.6. 関数

GAP の言語で書かれたプログラムを関数という. 関数を GAP のオブジェクトに適用すると, 新しいオブジェクトが返される. GAP における関数は数学的な意味の関数を含む、はるかに一般的なものである. 関数の例としては, 例えば Factorial( ) がある. Factorial(n) は非負整数 n に対して n の階乗 $n!=1\cdot 2\cdots(n-2)(n-1)n$ を与える.

```
gap> Factorial(50);
30414093201713378043612608166064768844377641568960512000000000000
```

すなわち 50! の値を一瞬で答える. 関数には常に ( ) が伴い, そこにいくつかのオブジェクトを書くことにより, 関数の答 (オブジェクト）が返って来る. 例えば, Gcd(,) は２つの整数の最大公約数を与える.

```
gap> Gcd(1234, 5678);
2
```

関数によってはオブジェクトを返す代わりに、何らかの動作をするものもある. 例えば, 関数 Print( ) は計算結果を画面に打ち出させる時に使われる.

```
gap> Print(1234, "\n");
1234
```

ここで後ろの "\n" はデータを画面にプリントした後の改行の命令である. これがないと以下のように計算結果と同じ行に gap> のプロンプトが現れてしまう.

```
gap> Print(1234);
1234gap>
```

自分で関数（この場合は一変数の関数）を定義する手軽な方法は写像（対応）を表す矢印 -> を使うものである. 例えば関数 $f(x) = x^3$ を与える関数 f = cubed を定義するには, cubed:= x -> x^3; を打ち込む (“cubed” は立方の意味).

```
gap> cubed:= x -> x^3;
function( x ) ... end
```

GAP は「関数 function(x) が定義できました. よろしく」と答える (2行目). 以後, 3 乗は cubed(x) で計算できる.

```
gap> cubed(5);
125
gap> cubed(37);
50653
```

より複雑な関数を定義する方法については後に説明する.

2 章　リスト と レコード

群や環を扱う基礎になるのは集合であり, そもそもコンピュータで集合をどのように表現するかというのが基本的な問題になる. GAP では以下に説明するように, 集合をリストの特別な場合として扱う. そのため, リストの考え方とその使い方に慣れることが要求される.

2.1 簡単なリスト

コンマで区切られ, 4 角括弧でまとめられたオブジェクトの集まりを**リスト**という. たとえば, 素数を小さい順に 10 個ならべたものを primes と名前をつけてリストにする.

```
gap> primes:= [2, 3, 5, 7, 11, 13, 17, 19, 23, 29];
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

リストはデータが一定順序でならんだ表（リスト）だと思っていいが、そこに新しいデータを付け加える 2 通りの関数がある. 一つはリストの最後にリストをつなげるもので関数 Append が使われる. たとえば, 29 の次の素数は 31 と 37 であるが, primes の後ろにリスト [31, 37] をつけ加えて,

```
gap> Append(primes, [31, 37]);
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 ]
```

関数 Append はリスト primes にデータ 31, 37 を加えたものを新しいリスト primes として返す. 一方, リストの最後に一つの要素をつけ加えるのには, 関数 Add を使う.

```
gap> Add(primes, 41);
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41 ]
```

リスト primes は数列 primes[n], あるいは整数 $1, 2, \ldots, n, \ldots$ 上の関数 primes[ ] とみることもできる. たとえば, primes の最初から 7 番目の項 17 は

```
gap> primes[7];
17
```

として得られる. primes[7] は値として扱うことができる. 例えば, 3 倍したり, 関数に代入したりできる.

```
gap> primes[7]*3;
51
gap> Factorial(primes[7]);
355687428096000
```

primes[ ] により, 指定された位置にデータを置くことができる. たとえば, 41 の次の素数は 43 なので, 43 はリストの最後の位置につけ加えられる. そこでまず関数 Length により, リストの最後の位置が何番目かを知り, その次の位置に 43 を入れる.

```
gap> Length(primes);
13
gap> primes[14]:= 43;
43
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43 ]
```

リストの途中を飛ばして, リストに値を代入することもできる. たとえば 20 番目の素数は 71 なので, primes[20] に 71 をしよう. すると

```
gap> primes[20]:= 71;
71
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,,,,,, 71 ]
gap> Length(primes);
20
```

となる. 余分なコンマが 5 個ならんでいるが, これは primes [15] から primes[19] までの 5 個に値が定義されていないことを示している. そして, primes は長さ 20 を持つリストになる.

CAP におけるリストは普通のプログラム言語の配列 (Array) に相当する. しかし配列があらかじめ n の上限 (配列の次元) を決めておかなければならないのに対し, リストは一度定義されてしまえば, どんな番号もとり得る. これは不精な人間にはおおいに助かるが, それ以上に素数の列 primes[n] のように無限に続く列を扱う場合に威力を発揮する.

しかし, ある番号での値を指定するためには, その前にリストが定義されていなければならないということに注意する. たとえば, リスト lll が定義されていない段階で 1 番目の値を指定すると

```
gap> lll[1]:= 2;
Variable: 'lll' must have a value
```

「リスト lll は値を持たない. 何を考えておるのか」と文句を云われる. これを避けるためには, まず空集合 [ ] としてリスト lll を定義しておく. 長さは 0 になる.

```
gap> lll:= [ ];
[ ]
gap> Length(lll);
0
```

その後は, 普通に値を取らせることができる.

```
gap> lll[1]:= 2;
2
```

リストに含まれるオブジェクトは, どんなタイプでもまた異なったタイプのものが混ざっても構わない. 例えば

```
gap> lll:= [true, "This is a String",,, 3];
[ true, "This is a String",,, 3 ]
```

またあるリストが他のリストの要素になることもできる. もっと, とんでもないことには, リスト自身をそのリストの要素として入れることもできる (再帰リスト).

```
gap> lll[3]:= [4,5,6];; lll;
[ true, "This is a String", [ 4, 5, 6 ],, 3 ]
gap> lll[4]:= lll;
[ true, "This is a String", [ 4, 5, 6 ], ~, 3 ]
```

2 番目の命令はリスト lll の 4 番目の位置にリスト lll 自身を代入せよというものであり, 結果は最後の式のようになる. ここで 4 番目の tilde ~ は, そこに全体と同じものが入っていることを示している. これはちょっと分かりにくいが, 例えば落語に出てくる, 頭のてっぺんにくぼみがあってそこに雨がたまり, やがて池ができ, その自分の頭の池に飛び込んで自殺する男の話を思い起こせば, 理解も深まるだろう. あるいはドラゴンボールで, 悟空とベジータが魔人ブーの体内で暴れている所にブー自身も登場する場面を思い出すのも良い. そこにブーが現れたとすると, その現れたブーの体内にも悟空とベジータ

がいるはずで, そこへまたブーが現れることになり, ... とこれもまた再帰リストである. このように再帰リストは我々の身近なところに転がっている.

”This is a String” のように文章の両端を ” で囲ったものを**ストリング (ひも)** という. ストリングは次のようにして, 特殊なリスト (途中に穴のない, つまりどんな番号にも値の入っている, 文字記号のみからなるリスト) として扱われる.

```
gap> s1:= ['H', 'a', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'];
"Hallo world"
gap> s1 = "Hallo world";
true
gap> s1[7];
'w'
```

リストが与えられたとき, その**部分リスト**を項の番号を中括弧 { } で指定することにより定義する.

```
gap> s1 := lll{[1,2,3]};
[ true, "This is a String", [ 4, 5, 6 ] ]
gap> s1{[2,3]} := ["New String", false];
[ "New String", false ]
gap> s1;
[ true, "New String", false ]
```

最初の指令は, リスト lll の最初の 3 項からなる部分リストを s1 とせよ, ということであり, 次は, リスト s1 の 2番目と 3 番目を新しいデータで置き換えよということである.

2.2 リストのコピー

リストが与えられているとき, 同じリストを作るには別の名前のリストに前のリストを代入すればよい.

```
gap> numbers := primes;; numbers = primes;
true
```

これで新しい numbers というリストに, primes のデータがすべて入ったことになる. ところが, こうして得られたリスト numbers はもとのリスト primes と一身同体, 切っても切れない関係になっている. (数学用語では inseparable = 別れられない = 非分離的という) になっている. primes の値を変えれば, numbers もそれに応じて変わり, numbers を変えれば primes も変わる. コピーを作るのは大抵もとのデータを利用して新しい表を作るような場合だから, そのときに元の表まで変わってしまっては, すこぶる不便である.

```
gap> numbers[3]:= 100;; numbers = primes
true
gap> primes[3];
100
```

このような事態が起こるのは, 以下の事情による. そもそもリスト primes を定義するということは, コンピュータのメモリーのある番地に名前 (primes さん) をつけて, primes 関係のデータはすべてそこに保存しておくということである. ところが, numbers := primes とすると, 新しい番地は用意されず numbers さんの住所も primes さんと同じにされてしまう. これでは, 2 人が同じ銀行口座を持っているようなものだから, numbers さんがお金を使うと primes さんの貯金も減ってしまうというわけである.

独立した家を持つにはどうしたらいいだろうか. 実は ShallowCopy という関数が用意されている. 深い関係になると離れられないから, 浅いおつきあいにしましょうね, ということでしょうか.

```
gap> primes[3]:= 5;; primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,,,,,, 71 ]
gap> numbers:= ShallowCopy(primes);; numbers = primes;
true
gap> numbers[3] := 100;; numbers = primes;
false
gap> primes[3];
5
```

ShallowCopy によって生み出された新しいリストは, もとのリストと別の番地を持ち, 独立なリストを作る. primes さんのもとを離れた numbers さんは, かくして新しい人生の一歩を踏み出すことになる.

2.3 集合

GAP では, 穴のないリスト (これを密なリストと云う) で, 重複がなく, 決められた順番に並んでいるものを**集合** という. ただし, この場合リストを構成してるデータが同じ性質のものである必要がある. 例えば成分が数の場合, [2, 3, 2, 5, 4] はリストであるが、これを集合としてみると, 重複を除き小さい順に並べて [2, 3, 4, 5] が集合になる. GAP には, リストが重複なく小さい順に並んでいる (strictly sorted list) かどうかを判定する関数 IsSSortedList がある. これが正しい時, リストは集合として扱われる. また与えられたリストから集合を作る関数 Set がある.

```
gap> number := [ 2, 3, 2, 5, 4];
gap> IsSSortedList(number);
false
gap> number := Set(number);
[ 2, 3, 4, 5 ]
gap> IsSSortedList(number);
true
```

関数 Set(number) はリスト number を変えないことに注意しておく. Set としての number を得るには, 上記のように置き直さなければならない.

上と同様のことが, アルファベットのストリングをデータとするリストについても成立する. この場合, 順番はアルファベットに関する辞書式順序である.

```
gap> animals := ["tiger", "puma", "giraffe", "panther", "tiger" ];
[ "tiger", "puma", "giraffe", "panther", "tiger" ]
gap> animals := Set(animals);
[ "giraffe", "panther", "puma", "tiger" ]
gap>IsSSortedList;
true
```

一方, ある元が集合に含まれるかどうかは `in` により判定できる.

```
gap> "giraffe" in animals;
true
gap> "jackal" in animals;
false
```

集合に新しい元を付け加えるには, 関数 `AddSet` を使う. `AddSet` はその元が集合に含まれているかどうかを判定し, 含まれていなければ, その元を順番にしたがって適正な位置に挿入した集合を返す. もし, その元がすでに含まれていれば, もとと同じ集合を返す.

```
gap> AddSet(animals, "jackal");
gap> animals;
[ "giraffe", "jackal", "panther", "puma", "tiger" ]
gap> AddSet(animals, "panther");
gap> animals;
[ "giraffe", "jackal", "panther", "puma", "tiger" ]
```

`"jackal"` がちゃんと 2 番目に入っていることに注目しよう. 3 番目の命令では, `"panther"`はすでに含まれているので集合 `animals` は変化しない.

集合が定義できると, 次は集合算であろう. GAP はこれも苦もなくこなす. GAP には関数 `Intersection` (共通部分) と `Union`(和集合) が用意されている.

```
gap> wildcats := ["tiger", "panther", "lion", "cheetah" ];
[ "tiger", "panther", "lion", "cheetah" ]
gap> Intersection(animals, wildcats);
[ "panther", "tiger" ]
gap> Union(animals, wildcats);
[ "cheetah", "giraffe", "jackal", "lion", "panther", "puma", "tiger" ]
```

`wildcats` は集合ではなくリストである（順番がそろっていない）. それにも拘らず, 関数 `Intersection` や `Union` の返す値は集合（順番がそろっている) になっていることに注意する. これは表の整理をするとき上手に使えば役に立つだろう.

2.4 変域

GAP では, **変域** (Range) とは整数のなす (有限) 等差数列であり, 特別なタイプのリストに他ならない. この等差数列を表わすのに, リストとしてすべての元を書く代わりに, [初項，第 2 項 .. 最終項] と書

く簡便法がある. 変域は次の節で説明するループ演算との関連で重要であるが, 変域の典型的な例としては, 1 から始まり, $1, 2, \cdots, 1000$ と 1 ずつ増加して, 特定の数 1000 まで進む等差数列である. このように, 等差が 1 の場合には, 第２項を省略して, `[1..1000]` と簡便に表す. 以下に例をあげる.

```
gap> [1..99999];
[ 1 .. 99999 ]
gap> [1,3..99999];)
[ 1, 3 .. 99999 ]
gap> Length (last);
50000
gap> [99999,99997..1];
[ 99999, 99997 .. 1 ]
```

1 番目は等差 1 の変域, 2 番目は等差 2, 最後は等差 -2 の変域である. ここで, `Length(last)` と書いたのは, すぐ前の値 (=last) を関数 `Length` に代入せよという意味で. 前の値を定義することなく参照できるので便利な記法である.

上に説明した変域 `[1..9999]` は, あくまで記法であるから, 変域自体は `[1, 2, 3 .. 999]` という形のリストに過ぎない. GAP には, 与えられたリストが変域であるかどうかを判定する関数 `IsRange` がある. また変域がリストとして与えられた場合にその表記を上のような簡便な形に表す関数 `ConvertToRangeRep` がある.

```
gap> a:= [-2,-1, 0, 1, 2, 3, 4, 5];
[ -2, -1, 0, 1, 2, 3, 4, 5 ]
gap> IsRange(a);
true
gap> ConvertToRangeRep( a );; a;
[ -2 .. 5 ]
gap> a[1] := 0;; IsRange(a );
false
```

2.5 For ループと While ループ

for ループと While ループ を上手に使いこなすことがコンピュータによる計算の決め手になる. まず, for ループから説明しよう. `pp` を置換からなるリストとし, そのリストに含まれているすべての置換を掛け合わせることを考える. GAP では for ループを使って次のようにやる.

```
gap> pp:= [(1,3,2,6,8)(4,5,9), (1,6)(2,7,8), (1,5,7)(2,3,8,6),
>          (1,8,9)(2,3,5,6,4), (1,9,8,6,3,4,7,2)];;
gap> prod:= ();
()
gap> for p in pp do
>       prod:= prod*p;
>    od;
gap> prod;
```

```
(1,8,4,2,3,6,5,9)
```

1 行目の命令でリスト pp を定義している．一行に書き切れないときは, return key を押して次の行以下に書いていけばよい．セミコロン ; を打ち込まない限り, GAP は唯 > を返すだけで、一続きの命令だと解釈する．リストのデータである置換はすべて巡回置換の積で表されていることに注意する．さて変数 prod を使って積を計算する．初期値として恒等置換 ( ) を入れておく．5 行目から 7 行目までが for ループである．p をループの変数とし, リスト pp に含まれる元を小さい順に一つ一つ取りだし, 変数 prod に右からかけていく．do がループの開始命令で, do を逆にした od; がループの終りを意味する．コンピュータは do と od; の間を p の条件にしたがって, ぐるぐる回ることになる．

For ループの一般的な書式は,

```
for 変数 in リスト do 命令 od;
```

である．for ループにより, リストに含めれるすべての元に対して命令が実行される (リストに穴があっても, そこは飛ばしてやってくれる．後で分かるように, これがまあ実に気の効いた配慮なのである)．ただし, for ループは実行するだけで値は返さないから, あらかじめ計算結果を記録する変数や関数を用意しておかなければならない．先ほどの例では 変数 -> p, リスト -> pp, 命令 -> prod := prod*p; であって, 計算結果が変数 prod に書き込まれた．

一般のプログラム言語では, for ループは次の形を取るのが普通である．

```
for 変数 from 初項 to 最終項 do 命令 od;
```

GAP の場合, これは単に「変域」という特別なリストを考えているのに過ぎない．2.4 節で述べた, 変域を表す簡単な記法にしたがえば, この特別な場合は,

```
for 変数 in [初項 .. 最終項] do 命令 od;
```

と書くことができる．例えば, 50! を for ループを使って計算するには以下のようにする．

```
gap> ff:= 1;
1
gap> for i in [1..50] do
>          ff:= ff*i;
>    od;
gap> ff;
30414093201713378043612608166064768844377641568960512000000000000
```

ここで, for ループを使って 1000 以下の素数のリスト primes を作るプログラムを紹介しておく．

```
gap> primes := [];;
gap> numbers := [2 .. 1000];;
gap> for p in numbers do
>      Add(primes, p);
>      for n in numbers do
>         if n mod p = 0 then
>         Unbind(numbers[n-1]);
```

```
>         fi;
>       od;
>    od;
```

ここで使っている方法は「エラトステネスのふるい」である. 2 から 1000 までの数のリスト numbers を用意する. まず 2 は素数だから, 素数のリスト primes の最初におく. 次に numbers から 2 で割れる数を全部排除する. 残っている数で 2 の次は 3. そこで 3 を primes のリストに加え, numbers から 3 で割れる数を全部排除する. numbers に残っている次の数は 5. そこで 5 を primes に加え, ... この操作を繰り返すことにより素数のリスト primes が得られる. プログラムの中に if が出てきたが, これは fi: と組になって条件文「if 命令」を構成する. より複雑な if 命令については次章 3.2 節で解説する.

次に while ループについて説明する. while ループの書き方は

`while` 条件 `do` 命令 `od;`

の形をとる. While ループは条件がみたされる限り, 命令の上をループで回り続ける. for ループと同様に while ループは od; により終了する.

次の例は, 与えられた整数の約数となる素数をすべて求めるプログラムである. ここでは, 素数の表 primes を利用して計算する. for ループと while ループがうまく組み合わされているのに注目して欲しい.

```
gap> n := 1333;;
gap> factors := [];;
gap> for p in primes do
>         while n mod p = 0 do
>              n := n/p;
>              Add(factors, p);
>         od;
>    od;
gap> factors;
[ 31, 43 ]
gap> n;
1
```

素数の表がある限り, n := 1333 を変えることによりどんな数 n に対しても素因数が決定できる. while ループは素数 p を固定したとき, その数が p で何回割れるか, すなわち n の素因数に p が何個現れるかを計算するのに使われている.

(注) GAP には上で作った命令 factors と同じように, 自然数 n の因数分解を与える関数 Factors(n) が既に定義されている.

```
gap> Factors(20160);
[ 2, 2, 2, 2, 2, 2, 3, 3, 5, 7 ]
```

素数の表を利用するもう一つの例として, 双子素数を求めるプログラムを載せておく. p と $p+2$ が共に素数のとき, $(p, p+2)$ を双子素数という. 双子素数は無限個存在すると予想されているが, まだ証明はされていない.

```
gap> twins := [];;
gap> for p in primes do
>          if p+2 in primes then
>               Add(twins, [p, p+2]);
>          fi;
>    od;
gap> twins;
[ [ 3, 5 ], [ 5, 7 ], [ 11, 13 ], [ 17, 19 ], [ 29, 31 ], [ 41, 43 ],
[ 59, 61 ], [ 71, 73 ], [ 101, 103 ], [ 107, 109 ], [ 137, 139 ],
[ 149, 151 ], [ 179, 181 ], [ 191, 193 ], [ 197, 199 ] ]
gap> Length(twins);
15
```

この例では, 双子素数の表 twins はリスト [p, p+2] を成分とするリストになっている. また後半のリストは, primes を 200 以下の素数として, twins を計算させた結果である. 200 以下の素数に対しては, 15 個の双子素数が存在することが分かる.

さらに, $p, p+6, p+12$ が素数のとき, $(p, p+6, p+12)$ を三子素数ということにすると, 三子素数は次のように計算される.

```
gap> trios := [];;
gap> for p in primes do
   >     if p+6 in primes then
   >        if p+12 in primes then
   >           Add(trios, [p, p+6, p+12]);
   >        fi;
   >     fi;
   >   od;
gap> trios;
[ [ 5, 11, 17 ], [ 7, 13, 19 ], [ 11, 17, 23 ], [ 17, 23, 29 ],
[ 31, 37, 43 ], [ 41, 47, 53 ], [ 47, 53, 59 ], [ 61, 67, 73 ],
[ 67, 73, 79 ], [ 97, 103, 109 ], [ 101, 107, 113 ],
[ 151, 157, 163 ], [ 167, 173, 179 ] ]
gap> Length(trios);
13
```

2.6 リストに対する操作

2.5 節で置換からなるリスト pp を定義し, pp に含まれる元の積を計算した. 実は GAP には, リストに含まれる元の積を計算する関数 Product が既に用意されている. これを用いると,

```
gap> Product(pp);
(1,8,4,2,3,6,5,9)
gap> Product([1..15]);
1307674368000
```

2 番目の例は整数のリスト `[1..15]` に関数 `Product` を適用している．これで, $15! = 1307674368000$ が計算できる．一般に関数 `Product` は, そのリストの元に積が定義されている限り, リストの元すべてにわたる積を計算するのである．

一方, リストの元に和が定義されていれば, 同様の関数 `Sum` が使える．例えば, $1+2+\cdots+100 = 5050$ は以下のように書ける．

```
gap> Sum([1..100]);
5050
```

他によく役に立つ関数として `List` がある．関数 `List` はリストと関数を変数として持ち, その関数をリストの元に作用させて得られるリストを値として返す．例えば, 以前定義したように `cubed` を関数 $f(x) = x^3$ とすると

```
gap> cubed:= x -> x^3;;
gap> List([2..10], cubed);
[ 8, 27, 64, 125, 216, 343, 512, 729, 1000 ]
```

すなわち, 2 から 10 までの数 x に対し, x^3 を並べたリストが得られる．関数 `Product` や `Sum` は, 変数として, リスト `[2..10]` と関数 `cubed` を取ることもできる．

```
gap> Product([2..10], cubed);
47784725839872000000
gap> Sum([2..10], cubed);
3024
```

それぞれ, 関数 `cubed` から得られたリスト `[ 8, 27, ..., 729, 1000 ]` に対する積と和を計算している．

素数のリスト `primes` が例えば 1000 以下の素数に対して定義されているとして, その中から一定の条件にあう元を取り出して新たにリストを作りたい．このようなとき関数 `Filtered` が使える．関数 `Filtered` はリストと, 元を取り出す性質を与える関数とを, 変数として持つ．例えば, 30 以下の素数を取り出したければ, 関数 `x -> x < 30` を使う．

```
gap> Filtered(primes, x -> x < 30);
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

以前述べた部分リストを作る操作`{ }` を思いだそう．この操作はリストの位置を変数として, その位置におけるリストの元を値として返す．

```
gap> primes{[1..10]};
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

2.7 ベクトルと行列

GAP では, **行ベクトル** とは同種の元からなる密なリストを意味する. (記法上, 列ベクトルよりは行ベクトルが基本になる. 列ベクトルは次に説明する行列の特殊な場合として扱う.)

```
gap> v := [3, 6, 2, 5/2];; IsRowVector(v);
true
```

`IsRowVector( )` は与えられたリストが行ベクトルかどうかを判定する関数である. GAP は行ベクトルに対して, 加法, スカラー倍, 内積を計算する.

```
gap> u := [1, 3, -2, 4];;
gap> v + u;
[ 4, 9, 0, 13/2 ]
gap> 2 * v; v * 1/3;
[ 6, 12, 4, 5 ]
[ 1, 2, 2/3, 5/6 ]
gap> v * u;
27
```

次に行列を定義する. GAP では同じ長さを持ったリストを元とする密なリストを**行列** という. たとえば, 次は 3 行 3 列の行列を定義する.

```
gap> m := [[1,-1, 1],
>          [2, 0,-1],
>          [1, 1, 1]];
[ [ 1, -1, 1 ], [ 2, 0, -1 ], [ 1, 1, 1 ] ]
gap> m[2][1];
2
```

行列 `m` の (i, j) 成分は `m[i][j]` で与えられる. たとえば, 行列 $m = (m_{ij})$ の (2,1) 成分 m_{21} は `m[2][1] = 2` となる. 通常のように, 行列は加法, スカラー倍, 乗法を持つ. ただし, 当然ながらサイズがあわなければ演算はできない.

```
gap> n := [[3, 4, 6],
>          [0,-2, 1],
>          [2, 4, 1],
>          [1, 0, 1]];
[ [ 3, 4, 6 ], [ 0, -2, 1 ], [ 2, 4, 1 ], [ 1, 0, 1 ] ]
gap> n * m;
[ [ 17, 3, 5 ], [ -3, 1, 3 ], [ 11, -1, -1 ], [ 2, 0, 2 ] ]
gap> m * n;
Vector *: <right> must have the same length as <left> (3)
```

`m * n` の計算では, 右と左のサイズが合わないというのでつむじを曲げられ, ブレイクループに逃げ込まれてしまう. 一方行列とベクトルの演算も可能である. しかしこの場合, ちょっと奇妙なことも起こる.

```
gap> [1,0,0] * m;
[ 1, -1, 1 ]
gap> m * [1, 0, 0];
[ 1, 2, 1 ]
```

最初の例は普通の行ベクトルと行列の計算だが, 2 番目の例では普通, この順番で行列と行ベクトルをかけることはできない. 実は GAP は `[1, 0, 0]` を列ベクトルとみなして, 行列との積を計算してくれるのである. (有難いことではあるが, 試験でこういう答案を書いたら, 先生は GAP 程やさしくありません).

GAP は整数計算以上に有限体での計算を得意としている. たとえば, 素数 p に対し整数 $\mathbb{Z}$ を $p\mathbb{Z}$ で割った剰余環 $\mathbb{Z}/p\mathbb{Z}$ は体になる. これを有限体, あるいは最初の発見者ガロアにちなんでガロア体, といい, CAP では `GF(p)` と表す. `GF` はガロア体 (Galois Filed) の略である. たとえば, `GF(5)` は整数を 5 で割った余り $\{0,1,2,3,4\}$ に `mod 5` で積と和を定義したものである. より一般に `GF(p)` の有限次拡大 (n 次拡大) は $q = p^n$ 個の元を持つ有限体 `GF(q)` を定義する. 有限体 $GF(q)$ の乗法群 $GF(q) - \{1\} = GF(q)^*$ は位数 $q-1$ の巡回群になり, その生成元を GAP では `Z(q)` と表す. (q が素数 p の場合, `Z(q)` は素数 p の原始根と呼ばれる.)

整数を成分とする行列やベクトルに有限体 `GF(q)` の元 `Z(q)^i` をスカラーとしてかけることにより, 行列やベクトルを有限体を成分とする行列やベクトルに変換し, そこで計算を進めるようにすることができる.

```
gap> m * One(GF(5));
[ [ Z(5)^0, Z(5)^2, Z(5)^0 ], [ Z(5), 0*Z(5), Z(5)^2 ],
[ Z(5)^0, Z(5)^0, Z(5)^0 ] ]
```

ここで `One(GF(5))` は有限体 `GF(5)` の単位元 1 を表す. $GF(5) = \{0,1,2,3,4\}$ とすれば, 要するに 1 であるが, これをかけることにより, $\mathbb{Z}$ から $\mathbb{Z}/5\mathbb{Z}$ に移行することになる. 5 の原始根は 2 であるから `Z(5) = 2` と思ってよい. この行列を $\{0,1,2,3,4\}$ の中で表現したければ, 関数 `Display` を使う.

```
gap> Display(m * One(GF(5));
1 4 1
2 . 4
1 1 1
```

これがもとの行列 `m` の各成分を `mod 5` で取った有限体 `GF(5)` 上の行列になる (ドット `.` はゼロを意味する). 直接 `Z(2)` や `Z(4)` をかけると,

```
gap> Display(m^2 * Z(2) + m * Z(4));
z = Z(4)
z^1 z^1 z^2
  1   1 z^2
z^1 z^1 z^2
```

親切なことには, Z(4) を z で置き換えて, z の巾として求める行列を書き下してくれるのである.

与えられた行列から小行列を作る操作は, リストから部分リストを作る方法で, 以下のように簡単に行なわれる.

```
gap> sm := m{ [1, 2]}{[2, 3]};
[ [ -1, 1 ], [ 0, -1 ] ]
gap> sm{ [1, 2]}{[2]} := [[-2], [0]];; sm;
[ [ -1, -2 ], [ 0, 0 ] ]
```

最初の例では, 3 次行列 m から 1 行目, 2 行目と 2 列目, 3 列目を選んで作った 2 次の小行列 sm を求めている. 2 番目の例ではこうして作った 2 次行列 sm の 2 列目を列ベクトル [[-2], [0]] で置き換えた行列を sm として書き出している.

群論に関係した関数として, GAP は与えられた群の元の位数を計算する関数 Order( ) を持っている. 有限体の元を成分とする行列の位数は有限であり次のようになる.

```
gap> Order(m * One(GF(5)));
8
gap> Order(m);
infinity
```

最初の例は, 行列 A = m * One(GF(5)) の位数が 8, つまり, $A^m = I$ となる最小の数が 8 であることを示している. 次の例では整数行列 m の位数が無限大であることを示している.

以下に行列に関する種々の関数をあげておく. 行列 A の転置行列 ${}^tA$ を求めるには, 関数 TransposedMat を使う.

```
gap> TransposedMat( [[ 1, 2], [3, 4]]);
[ [ 1, 3 ], [ 2, 4 ] ]
gap> TransposedMat ( [ [1..5]]);
[ [ 1 ], [ 2 ], [ 3 ], [ 4 ], [ 5 ] ]
```

2 番目の例では, [[1..5]] は [1, 2, 3, 4, 5] を行ベクトルとする 1 行 5 列の行列を表す. したがってその転置行列は 5 行 1 列の行列になる.

n 次単位行列 I_n は関数 IdentityMat(n) で表される. また, m 行 n 列の零行列は関数 NullMat(m,n) で与えられる.

```
gap> IdentityMat(3);
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ]
gap> NullMat(2,3);
[ [ 0, 0, 0 ], [ 0, 0, 0 ] ]
```

正方行列 $A = (a_{ij})$ に対し, $\mathrm{Tr}\ A = \sum_i a_{ii}$ を A のトレース (trace) という. $\mathrm{Tr}\ A$ は関数 TraceMat で計算できる.

```
gap> TraceMat([[1,2,3], [4,5,6], [7,8,9]]);
15
gap> I:= IdentityMat(3);;
gap> TraceMat(I);
3
```

正方行列 A の行列式 $\det A$ および 行列 A の階数 $r(A)$ はそれぞれ関数 DeterminantMat, RankMat で計算できる.

```
gap> a:= [[1,2,-1], [2, 5, 0], [3, 3, 1]];
[ [ 1, 2, -1 ], [ 2, 5, 0 ], [ 3, 3, 1 ] ]
gap> DeterminantMat(a);
10
gap> b:= [[4,1,2], [3,-1,4], [-1,-2,2]];;
gap> RankMat(b);
2
```

しかし, 行列式や階数の計算は基本的に行列の基本変形に基づいているので, 大きなサイズの行列の計算には時間がかかる.

正則行列 A に対し, A の逆行列 A^{-1} は次のように書けばよい.

```
gap> a^-1;
[ [ 1/2, -1/2, 1/2 ], [ -1/5, 2/5, -1/5 ], [ -9/10, 3/10, 1/10 ] ]
```

2.8 簡単なレコード

GAP では幾つかのデータを複合的に記録させておく手段として, **レコード** (記録) が用意されている. リストと同じように, レコードも部分オブジェクト (レコード成分という) からできている. しかしリストの部分オブジェクトが primes[1], primes[2] のように番号によって与えられるのと異なり, レコードでは, 各成分が直接名前で参照される. 例えば日付けに関するデータを保存しておくには

```
gap> date := rec(year := 2003,
>                month := "June",
>                day := 14);
rec( year := 2003, month := "June", day := 14 )
```

とする. これで, year, month, day を成分とするレコードが定義された. レコードの各成分を参照するには,

```
gap> date.year; date.month; date.day;
2003
"June"
14
```

とすればよい. レコードの中に, さらに詳しいデータをレコードとして付け加えることもできる.

```
gap> date.time := rec(hour:= 19, minute := 23, second := 12);
rec( hour := 19, minute := 23, second := 12 )
gap> date;
rec( year := 2003, month := "June", day := 14,
     time := rec( hour := 19, minute := 23, second := 12 ) )
```

このように, どんどん新しいデータを付け加えてレコードを完成させていくわけだが、時としてレコードの中にどんな項目があったか忘れてしまう. そんな場合は, 関数 RecNames によりレコードの各成分を参照できる.

```
gap> RecNames(date);
[ "year", "month", "day", "time" ]
```

例えば誕生日のデータを集めて保存しておくような場合に, レコードは有効である.

3 章　さらに関数について

3.1 関数の書き方

画面に hello, world. と文字を書く関数は次のように定義される.

```
gap> sayhello:= function()
>  Print("hello, world. \n" );
>  end;
function ( ) ... end
```

GAP の関数はアラジンの魔法のランプのようなもので, 呼ばれると大男が出てきて命令を実行し, 終るとランプに戻っていく. 多くの場合関数の値を返すが, 上の例では Print 命令を実行するのみである. すなわち 2 行目で hello world と画面に書き, 次いで改行 “\n” する.

関数は次の書式で書かれる.

```
function ( 変数 ) 命令 end
```

まず, function で関数を宣言し, その後の ( ) の中に関数の変数（パラメータ）を, コンマで区切って書く. 変数の数は上のように 0 個でもいいし, 何個でも許される. function( ) の後ろにはセミコロンを書かない. プログラムは最後に end; と書くことにより終了する. 定義した関数 sayhello の中身を見るには次のようにする.

```
gap> Print(sayhello, "\n");
function (  )
    Print( "hello world. \n" );
    return;
end
```

sayhello の後ろに改行指令 "\n" をつけないと, endgap> のように, 最後の end と次の GAP のプロンプト gap> がつながってしまう.

新しく定義された関数 sayhello を実行するには sayhello(); と書けばよい.

```
gap> sayhello();
hello world.
```

3.2 If 構文

次の例は整数の正負を判定する符合関数を定義するプログラムである.

```
gap> sign := function (n)
>           if n < 0 then
>              return -1;
>           elif n = 0 then
>              return 0;
>           else
>              return 1;
>           fi;
>     end;
function( n ) ... end
gap> sign(0); sign(-99); sign(11);
0
-1
1
```

符合関数 sign(n) は整数 n が 正, 負 または 0 にしたがって値 $1, -1, 0$ を返す. ここで使われているのが if 命令である. if 命令は次の書式で表され, 条件によって別れ道を指定する.

```
if  条件  then  命令  elif  命令  then  命令  else  命令  fi
```

elif はもちろん else if をつなげた造語で, 間にいくつ入れてもいいが最後は, else できちっと締める. 必要がなければ, 2.5 節の例のように elif も else も省略できる.

山のガイドブックの中に, もし山の中で熊に会ったらどうするかという話で, 熊を刺激しないように笑顔で話しかける, もしそれで駄目だったらそうーっと離れる, もしそれで駄目だったら必死になって逃げる, もしそれでも駄目だったら木に登る, もしそれでも駄目だったら川に飛び込む, もしそれでも駄目だったら死んだふりをする, ..., というのがあった. もし駄目だったらと何回も云っているところを見ると著者自身もこれで助かるとは思っていないようである. そもそも死んだふりをして助からなかった人の話は聞けないから最後の方法がどの程度有効かも分からないが, とにかくこれが典型的な if 構文である.

3.1 節で, 関数はランプの精のようなものだと云ったが, 関数を定義する過程は, 我々自身がアラビアンナイトの世界に飛び込むことを意味する. function(a, b, c) と関数が宣言された時点で, 我々はこちらの世界から変数 a, b, c を携えて別世界に行く. end 命令でプログラムが終了しもとの世界に戻れ

るが, プログラムの途中で宝物を抱えて逃げて来ることもできる. それが return 命令で, 上の例では, それぞれ -1, 0, 1 と共に現実世界に戻って来るのである.

2.1 節で再帰リストについて述べたが, CAP では関数を定義する文章の中でその関数自身を使うことができる. これは, にわとりが先か卵が先かという問題とも密接につながっているようだが, 漸化式で定義された数列を扱うときなどはすこぶる都合がよい. 以下の例はフィボナッチ数列を計算するプログラムである.

```
gap> fib:= function(n)
>       if n in [1,2] then
>             return 1;
>       else
>             return fib(n-1) + fib(n-2);
>       fi;
>    end;
function( n ) ... end
gap> fib(15); fib(20); fib(30);
610
6765
832040
```

ここでは, 初期条件 fib(1) = 1, fib(2) = 1 とフィボナッチ数列の漸化式 (5 行目) を与えるだけで, fib(n) をすべて計算してくれる. しかしプログラムは短くて書き易いけれど, 再帰命令の入ったプログラムのループの回数は増加する. そこでプログラムの効率, つまり計算時間は長くなってしまう. 効率の良い計算をするためには, たとえプログラムは長くなっても再帰命令ではなく, 繰り返し命令を使ったプログラムにした方がよい（次節参照）.

3.3 局所変数

次の例は 2 つの整数 a, b の最大公約数 gcd(a,b) をユークリッドの互除法を用いて計算するプログラムである.

```
gap> gcd:= function(a,b)
>            local c;
>            while b <> 0 do
>               c:= b;
>               b:= a mod b;
>               a:= c;
>            od;
>          return c;
>    end;
function( a, b ) ... end
gap> gcd (30,63);
3
```

a と b の最大公約数を求めるには, 組 (a,b) を a' = b と a を b で割った余り b' の組 (a', b') で置き換えるという操作を繰り返し, b' がゼロになったときの b が最大公約数を与える. これを実際に行なおうとすると, 計算の途中で b の値を退避させておくために, どうしてももう一つ変数 c が必要になる. この変数は, 関数の定義が書かれている別世界, つまりアラビアンナイトの世界でのみ意味を持つ変数である. これをプログラムの始めで local c; として宣言し, **局所変数** という. 局所変数は, あくまで gcd(a,b) の世界でのみ意味を持つので, 外の世界で同じ変数 c を使っても影響は及ばない. これは大事なことで, function の定義は隠されているか、自分で定義したとしても中身は覚えていないのが常であるので, 局所変数にしておけばこちらの世界で安心して好きな変数を使えるのである. 次の例で変数 c の値が変化しないことを見て欲しい.

```
gap> c:= 7;;
gap> gcd(30,63);
3
gap> c;
7
```

局所変数を使って, フィボナッチ数列の以前より効率の良い（しかし以前より複雑な）プログラムを作ることができる.

```
gap> fib := function (n)
>             local f1, f2, f3, i;
>             f1 := 1;  f2 := 1;
>             for  i  in [3..n] do
>                  f3 := f1 + f2;
>                  f1 := f2;
>                  f2 := f3;
>             od;
>             return f2;
>           end;
function( n ) ... end
gap> List([1..10], fib);
[ 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ]
```

4 章　ファイルを使いこなす

GAP で関数やプログラムを定義しても, 一度 GAP を終了すればこれらの関数やプログラムは消去されてしまい, 次に使うときはまた定義しなおさなければならない. また計算されたデータも終了と同時に消されてしまう. 一度定義した関数やプログラムを GAP に忘れずに記憶させ, 大量のデータを GAP に読み込ませるためにはファイルをうまく使わなければならない. 以下ファイルの使い方を説明する.

4.1 Read

例えば, 2.5 節で出てきた素数を計算するプログラムを保存しておきたいとする．その場合, 適当なエディターを使って, 次のようにファイル myfile.g を作る．(拡張子は別に.g でなくても良い).

```
###### myfile.g ######
primes := [];
numbers := [2 .. 5000];
for p in numbers do   ### prime numbers less than 5000
  Add(primes, p);
  for n in numbers do
    if n mod p = 0 then
    Unbind(numbers[n-1]);
    fi;
  od;
od;
```

ファイルの中では, 命令は実行されないので, 以前のように primes := [];; と 2 重のセミコロンにする必要はない．また GAP の中では # は**コメント文**の働きをする．各行で # 以下は無視されるので # を使ってプログラムにコメントをつけるのが普通である．

```
gap> Read("myfile.g");
```

とすることにより, myfile.g の内容が GAP に読み込まれる．特に, リスト primes には 5000 以下の素数の表が蓄えられる．プログラムを編集する場合, 例えば 5,000 を 10,000 にしたかったらファイル myfile.g を編集し直せば良い．これは GAP の中で編集作業をするよりは, はるかに簡単である．

なお, 自分用のファイル myfile.g を GAP の立ち上げ時に自動的に読み込ませることもできる．ホームディレクトリにファイル .gaprc を作り, ここにファイルの読み込みや, 良く使う定義を以下のように書いておけばよい．GAP は起動時に最初に.gaprc を読みにいくので, ここで必要なデータがすべて GAP に読み込まれる．例えば

```
Read("myfile.g");
a:= [[1,2,-1], [2, 5, 0], [3, 3, 1]];
b:= [[4,1,2], [3,-1,4], [-1,-2,2]];
```

4.2 PrintTo

4.1 節で計算した 5000 以下の素数のリストをデータとして保存しておくには, 関数 PrintTo を使う．1.6 節に出てきた関数 Print との違いは, 画面に打ち出す代わりにファイルに書き出すということである．関数 PrintTo は

PrintTo(filename, obj1, obj2 ...)

の書式を持つ．obj1, obj2, ... はファイルに書き出すオブジェクトを表す.

```
gap> PrintTo("primelist.g", primes);
```

これにより, リスト primes の内容が, ファイル primelist.g にデータとして保存される. 次に述べる関数 AppendTo との違いは, PrintTo では実行されるたびにデータが上書きされ, 以前のデータが消去されるのに対し, AppendTo では, 以前のデータの後ろに新しいデータが付け加えられていくことである.

4.3 AppendTo

AppendTo は指定したファイルにデータを順次付け加えて保存していく関数を表す. 以下の書式を持つ.

AppendTo(filename, obj1, obj2 ...)

5 章 GAP で群を調べよう

5.1 生成系と部分群

G を群, S を G の部分集合とする. S を含む G の最小の部分群を H を S で生成された部分群という. H は, S の元と, その逆元を好きなように, いくつか掛け合わせて出来る元の全体である. $H = G$ となる時, S を G の**生成系**, S の元を G の**生成元**という. 例えば, $G = S_n$ を n 次対称群とする. G の元は巡回置換の積として一意的に表される (1.4 参照). そこで S を S_n の巡回置換全体の集合とすると, S は G の生成系になる. 一方, 任意の巡回置換は互換の積として表される. 例えば,

$$(1,2,3,4,5) = (1,2)(1,3)(1,4)(1,5).$$

そこで S' を互換全体の集合とすると, S' も G の生成系になる. さらに,

$$S'' = \{(1,2),(2,3),\ldots,(n-1,n)\}$$

とおく. $(1,5) = (1,2)(2,3)(3,4)(4,5)(3,4)(2,3)(1,2)$ というようなことに注意すると, S'' も S_n の生成系になることが分かる. 明らかに,

$$S'' \subset S' \subset S$$

である. 群 G に関する性質の多くは, その生成元に対して成立することを確かめれば, 自動的に全体で成立するという場合が多い. その意味で, 生成系は元の個数が少ない程, 効率的である. しかし, 所詮, 生成系だけで全ての話がすむ訳でもなく, 群 G 全体について議論しようと思えば, G の元を S の元の積として具体的に書く必要もまた生ずるであろう. こうなると, 上の例からも分かるように S が小さいというのは存外に不便なことである. 過ぎたるは及ばざるが如し, 偏えに風の前の塵に同じ, ということである. もちろん, 目的にもよるのであるが, どれが本当に良い生成系かどうかは実は難しい問題である. 目先の利益に囚われて安逸に走ってはいけない

さて, S_n の生成元をさらに減らそうとすると, 1 個では無理だから, 最低限 2 個は必要である. 2 個で出来るだろうか. 実は出来るのである. S_n は $a = (1,2)$ と $b = (1,2,\ldots,n-1,n)$ の 2 個の元で生成される. 実際, 計算してみると,

$$b^{-1}ab = (2,3), \quad b^{-2}ab^2 = (3,4), \quad \ldots \quad b^{-n+2}ab^{n-2} = (n-1,n)$$

となることが分かる. したがって, a, b で生成された S_n の部分群は生成系 S'' を含み, S_n に一致する.

このような極端な例をもてあそぶことは, 過ぎたるは ... の格言の如く, 善良な市民を欺く邪教のようにも思えるのだが, 実はこれが意外と役に立つのである. それは GAP の本質に関係している. 確かに GAP は群の計算を気楽にやってくれる. 我々がある群に関するデータを与えれば, それに関して計算をし, 例えば部分群に関するデータを返して来る. しかしそれでは, どのようにしたら GAP に群 G を認識させることが出来るのだろうか. そして, GAP はどのようにして, その答えである部分群 H を我々に伝えることができるのだろうか. その鍵を握るのが生成系である. GAP は置換に関する演算を認識する. つまり, GAP の中には, 無限個の文字 $\{1, 2, 3, \dots\}$ に関する置換群 S_∞ が入っていると思ってもよい. そして, 有限群 G を有限個の置換により生成された S_∞ の部分群 (このような群を**置換群**という) として認識する. これが GAP の世界で群 G を定義することの意味である. 我々は, 生成系を指定することにより, GAP に群 G を伝え, GAP は計算の答えを, 生成元を与えることで我々に返して来るのである. 一度 群 G が定義されれば, 後は G を母体として種々の群操作が可能になる.

さて S_n を簡単に定義しようと思ったら, 2 つの元 $a = (1, 2)$ と $b = (1, 2, \dots, n)$ を生成元として与えればよい. 生成系 $\{a, b\}$ が理論的に有用かどうかは別として, GAP への伝達手段としては, これが最も効率的であろう. 実際にやってみると,

```
gap> s8 := Group((1,2), (1,2,3,4,5,6,7,8) );
Group([ (1,2), (1,2,3,4,5,6,7,8) ])
```

これにより, 8 文字 $\{1, 2, \dots, 8\}$ の置換群として, 対称群 S_8 が定義できた. 以後, 対称群 S_8 を扱うには, 単に `s8` を呼び出せばよい.

GAP では群 `s8` は集合として定義される. そこで群 `s8` の位数を知るには, 集合の元の個数を与える関数 `Size` を利用する. 以下のように $|S_8| = 8! = 40320$ が得られる.

```
gap> Size(s8);
40320
gap> Size(s8) = Factorial(8);
true
```

２番目の等式は, GAP に S_8 の位数が $8!$ に等しいかを尋ねている. 答えは 「正しい」. `Size` は, 一般に集合の個数を与える関数であるが, 群の場合, 位数 (order) を与える関数も使うことができる.

```
gap> Order(s8);
40320
```

さて G の部分群 H と K に対して, $H \cap K$ は, また G の部分群になる. そこで GAP では, 部分群 $H \cap K$ は単に `Intersection(H,K)` で定義できる.

```
gap> H := Group((1,2), (1,2,3,4));
Group([ (1,2), (1,2,3,4) ])
gap> K := Group((2,3), (3,4,5));
Group([ (2,3), (3,4,5) ])
gap> L := Intersection(H,K);
Group([ (2,3), (2,4,3) ])
```

```
gap> IsGroup(L);
true
```

IsGroup(L) は, 与えられた集合 L が群であるかどうかを判定する関数である. ところで, 和集合 $H \cup K$ はもちろん群ではない. H と K を含む最小の G の部分群, すなわち H と K で生成された部分群を GAP は ClosureGroup(H,K) として求めることができる.

```
gap> M := Union(H,K);;
gap> IsGroup(M);
false
gap> M_1 := ClosureGroup(H,K);
Group([ (1,2), (1,2,3,4), (3,4,5) ])
gap> IsGroup(M_1);
true
gap> M_2 := Group(Union(H,K));
<permutation group with 42 generators>
gap> M_1 = M_2;
true
```

上の式は, M_1 が確かに和集合 $H \cup K$ で生成される部分群に一致することを示している. しかし, M_2 よりも M_1 の方が効率の良い作り方であることに注意する. ひとたび和集合 $H \cup K$ を取ると, 群の構造は忘れ去られ, 単に 42 個の元からなる集合になってしまう. そこで M_2 は 42 個の元で生成された群となり, 生成元表示は難しくなるのである.

H を G の部分群とする. G の元 g に対し, $g^{-1}Hg = \{g^{-1}xg \mid x \in H\}$ は G の部分群になる. $g^{-1}Hg$ を H の**共役部分群** (conjugate subgroup) という. $g^{-1}Hg = H^g$ とも表すが, GAP でもこの記号で共役部分群を定義できる.

```
gap> H := Group((1,2), (1,2,3,4));
Group([ (1,2), (1,2,3,4) ])
gap> g := (1,4,2,3,6);
(1,4,2,3,6)
gap> H^g;
Group([ (3,4), (2,4,3,6) ])
```

H は, 元 $(1,2)$ と $(1,2,3,4)$ で生成された群であるが, 共役な部分群 H^g は, 元 $(3,4)$ と $(2,4,3,6)$ で生成された部分群になることが分かる.

さて, H とその部分群 L を以前定義された群とする. $g \in G$ に対し, L の共役部分群 L^g は必ずしも H の部分群にはならない. GAP では群 G と 群 H に対して, H が G の部分群になるかどうかを IsSubgroup(G,H) によって判定できる. ただし, 実用的には G も H も共通の群 Ω の部分群である場合が主なので, 単に $G \cap H$ が H に一致するかどうか確かめればすむことである.

```
gap> g := (1,2);
(1,2)
gap> IsSubgroup(H, L^g);
true
gap> g := (2,3,5);
(2,3,5)
gap> IsSubgroup(H, L^g);
false
```

5.2 巡回群とアーベル群

ただひとつの元で生成される群を**巡回群** (cyclic group) という. 巡回群は最も構造の簡単な群である. 例えば,

```
gap> H:= Group((1,2,3,4,5));
Group([ (1,2,3,4,5) ])
gap> Order(H);
5
```

H は 元 $(1,2,3,4,5)$ で生成される位数 5 の巡回群である. 一般に, 群 G の元 g に対して, g で生成される巡回群の位数, すなわち, $g^k = 1$ となる最小の正の整数を g の**位数**という. GAP に g の位数を聞くには, 同じ `Order` を使えばよい.

```
gap> g := (1,2,3,4,5);;
gap> Order(g);
5
```

群 G が巡回群かどうかを知るには, `IsCyclic(G)` を使う.

```
gap> IsCyclic(H);
true
```

これは, それほど役に立たないかも知れない. 実際 GAP では, 群を生成元で返して来るので, 生成元が一つなら, 改めて聞いてみるまでもなく, それは巡回群である. 以後, 位数 n の巡回群を $\mathbb{Z}/n\mathbb{Z}$ と表す. (この記号の意味は後に説明する.) 群の演算が可換な群, すなわち, 任意の $x, y \in G$ に対して, $xy = yx$ が成り立つとき, G を**アーベル群** (abelian group) または**可換群**という. 巡回群はアーベル群である. 生成元で与えられた群がアーベル群かどうかは, 一目で分かるというものではない. GAP では, `IsAbelian(G)` を使う. 例えば,

```
gap> elab := Group((1,2),(3,4));
Group([ (1,2), (3,4) ])
gap> Size(elab);
4
gap> IsAbelian(elab); IsCyclic(elab);
true
```

```
false
gap> IsAbelian(s8);
false
```

元 $(1,2),(3,4)$ で生成された群 K =elab は位数 4 のアーベル群であるが, S_8 はアーベル群でないことが分かる. また, K は巡回群ではないので, $K \simeq \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ である. K を**クラインの 4 元 群**という. 一般に, p を素数としたとき, 巡回群 $\mathbb{Z}/p\mathbb{Z}$ の r 個の直積 $(\mathbb{Z}/p\mathbb{Z})^r$ に同型な群を**基本アーベル (p-) 群** (elementary abelian group) という. GAP には与えられた群が基本アーベル群かどうかを判定する命令 IsElementaryAbelian がある.

```
gap> IsElementaryAbelian(elab);
true
```

K =elab は, 基本アーベル 2 群となり, 確かに $K \simeq \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ であることが分かった.

5.3 導来群と可解群

G を群とする. $xyx^{-1}y^{-1}$ $(x, y \in G)$ によって生成された G の部分群 G' を G の**導来群** (derived subgroup) または **交換子群** (commutator subgroup) と呼ぶ. 交換子群の名前は, xyx_1y^{-1} の形の元を交換子ということから来ている. $xyx^{-1}y^{-1} = [x, y]$ と書いてブラケット積と呼ぶことも多い. そこで, $G_1 = [G, G]$ と表すこともある. GAP には与えられた群の導来群を計算する関数 DerivedSubgroup が用意されている.

A_n を**交代群**, すなわち偶置換全体からなる S_n の部分群, とする. 以下では, 交代群 A_n が S_n の導来群になっていることを利用して, GAP に A_n を認識させることにしよう. まず, A_n が S_n の導来群に一致することを簡単に説明しておく. $G = S_n$ とおく. $x, y \in G$ に対し, 交換子 $xyx^{-1}y^{-1}$ は明らかに偶置換である. したがって $xyx^{-1}y^{-1} \in A_n$. 生成元が含まれるから $G' \subseteq A_n$ が分かる. 逆の包含関係 $A_n \subseteq G'$ を示す. そのため

(*) A_n は長さ 3 の巡回置換で生成される

ことを示そう. 長さ 3 の巡回置換は偶置換である. 例えば, (1,2,3) = (1,3)(2,3). A_n の元は偶置換だから, $(a,b)(c,d)$ の形の元で生成される. そこで, $(a,b)(c,d)$ が長さ 3 の巡回置換で生成されることを見ればよい. 集合 $\{a,b,c,d\}$ の元の個数が 3 個か 4 個かに応じて $(a,b)(c,d)$ は 2 種類のタイプに分かれる. 例えば, (1,2)(2,3) または (1,2)(4,5). 前者の場合, $(1,2)(2,3) = (1,3,2)$ は長さ 3 の巡回置換. 後者の場合, $(1,2)(4,5) = (1,2)(2,4)(2,4)(4,5) = (1,4,2)(2,5,4)$ でこれも O.K.

そこで, $A_n \subseteq G'$ を示すには, 長さ 3 の巡回置換が G' に含まれることを示せばよい. 例えば $g = (1,2,3)$ とすると, $g = (1,3,2)^2 = (1,2)(2,3)(1,2)(2,3) = xyx^{-1}y^{-1}$ と表される. ただし, $x = (1,2), y = (2,3)$. したがって, $g \in G'$ となり, $A_n \subset G'$ も成り立つ. 以上で $G' = A_n$ となることが分かった.

以上の議論から, 交代群 A_n を S_n の導来群として定義できるできることが分かった. よく考えて見れば, 交代群をどうやって定義するかはそんなに簡単ではない. 偶置換の全体というのを GAP に分からせるのは, かなり面倒なことである. 他にも手段がないわけではないが, 対称群の導来群というのが, 一番手っ取り早い方法であろう.

さて, 交代群 A_8 を定義しよう.

```
gap> a8 := DerivedSubgroup(s8);
Group([ (1,2,3), (2,3,4), (2,4)(3,5), (2,6,4), (2,4)(5,7),
         (2,8,6,4)(3,5) ])
gap> Size(a8);
20160
IsAbelian(a8);
false
```

交代群 A_8 が置換 $(1,2,3),(2,3,4),(2,4)(3,5),\cdots$ で生成された群として定義され, a8 と名前が付けられたのである. ここで 2 行目の表示は, あくまで生成元を表わしているのであって, a8 の元を全部書いているのではないことに注意する. A_8 の位数は 20160 であるから元を全部書いたら大変なことになる. 生成元による表示が如何に効率が良いか分かるだろう. 生成元を与えることにより群は完全に決定されるから, 集合 a8 を表示するにはこれで充分である. 表舞台には現れないが, コンピュータには全ての元がストックされており, 必要なときに必要な元を使ってコンピュータが計算を実行してくれるのである. 我々の目に見えないところで大変な努力がなされているのですね.

上の例からも分かるように, GAP は計算の結果得られて群を生成元によって表示して我々に返す. しかし, もちろん GAP の中には群自身に関する全ての情報が入っているのであって、必要があればそれを取り出すことができる. 実際, せっかく群ができたのであれば, 生成元などとけちくさいことを言わず, 全ての元を書き出して見たいという誘惑にかられるのも人情である. そんなときは関数 Elemnts( ) を使うことができる. しかし, ゆめゆめ関数 Elements を s8 のような位数の大きい群に対して試みようとしてはいけない. 悲惨な結果になること請合いである.

```
gap> s3:= Group((1,2), (1,2,3));
Group([ (1,2), (1,2,3) ])
gap> Elements(s3);
[ (), (2,3), (1,2), (1,2,3), (1,3,2), (1,3) ]
```

GAP は群 s3 を集合として扱うから, s3 の元は順番に従って並んでいる. 上の場合どういう規則によって順番が決っているのだろうか. それは上の各元を置換として表してみるとよく分かる.

$$\begin{pmatrix}1,2,3\\1,2,3\end{pmatrix},\ \begin{pmatrix}1,2,3\\1,3,2\end{pmatrix},\ \begin{pmatrix}1,2,3\\2,1,3\end{pmatrix},\ \begin{pmatrix}1,2,3\\2,3,1\end{pmatrix},\ \begin{pmatrix}1,2,3\\3,1,2\end{pmatrix},\ \begin{pmatrix}1,2,3\\3,2,1\end{pmatrix}.$$

この順番は数字 1,2,3 に関する辞書式順序である. この事実が GAP が群をどう理解しているかを良く表している. 人間にとって群は, その中で元がくっついたり, 離れたり, 分子が躍動している集合のイメージがあるが, GAP にとっては, 単に辞書式順序に並べられた置換の列に過ぎないのである. コンピュータの宿命とは言え, ちょっと寂しい気もする.

さて, 群 G に対して, その導来群 G' は G の正規部分群になる. ここで**正規部分群** (normal subgroup) の定義を復習しておこう. G の部分群 H が G の正規部分群とは $g^{-1}Hg = H$ が全ての $g \in G$ に対して成り立つことである. これは, $g^{-1}hg \in H$ が全ての $g \in G, h \in H$ に対して成り立つと言っても同じである. H が G の正規部分群の時, $H \triangleleft G$ と記す. GAP には与えられた部分群が 正規部分群になるかどうかを判定する関数 IsNormal( ) がある. 一応確認してみると,

```
gap> IsNormal(s8,a8);
true
gap> IsNormal(s8,s3);
false
```

確かに, A_8 は S_8 の正規部分群であることが分かった. ついでに, S_3 は S_8 の正規部分群ではない.

導来群の定義からすぐ分かるように, G がアーベル群の場合, 導来群 G' は単位群 $\{1\}$ になる. 実は, G がアーベル群であることと, 導来群が単位群になることとは同値である. つまらない例だけれど, クラインの 4 元群 `elab` で計算してみると,

```
gap> DerivedSubgroup(elab);
Group(())
```

これは, `elab` の導来群が $()$ で生成される群, すなわち単位群であることを示している.

G を有限群とするとき, G の導来群 $[G,G]$ を G_1 と表す. G の部分群の列

$$G = G_0 \supset G_1 \supset \cdots \supset G_n \supset \cdots$$

を $G_n = [G_{n-1}, G_{n-1}]$ により定義し, G の **導来列** (derived series) という. 各 G_i は G_{i-1} の正規部分群である. 有限群の重要なクラスとして**可解群**があるが, 導来列は可解群になるための判定条件を与える. G の 導来列が有限回の後, 単位群 $\{1\}$ に到達すること, つまりある $n \geq 1$ に対して $G_n = \{1\}$ となることが, G が可解になるための必要十分条件である.

GAP は与えられた群の導来列を計算する関数 `DerivedSeries( )` を持っている. 例えば, $G = S_4$ の導来列を調べてみると,

```
gap> s4:= Group((1,2),(1,2,3,4));;
gap> D := DerivedSeries(s4);
[ Group([ (3,4), (2,3,4), (1,2)(3,4), (1,4)(2,3) ]),
  Group([ (2,3,4), (1,2)(3,4), (1,4)(2,3) ]),
  Group([ (1,2)(3,4), (1,4)(2,3) ]), Group(()) ]
gap> List(last, Size);
[ 24, 12, 4, 1 ]
```

GAP は, 導来列に現れる群をリストとして返している. $D[1] = G_0, D[2] = G_1, D[3] = G_2, D[4] = G_3$ である. 前半で対称群 S_4 の導来列を計算し, 現われる部分群をすべて決定している. しかし, これではごちゃごちゃして見づらいので、後半は, 導来列に現れる群の位数を並べて、リストを作ってある. これは $G =$`s4` とするとき,

$$G = G_0 \supset G_1 \supset G_2 \supset G_3 = \{1\}$$

であって,

$$|G_0| = 24, \quad |G_1| = 12, \quad |G_2| = 4, \quad |G_3| = 1$$

を示している. したがって, $G = S_4$ は可解群である. 同時に, A_4 も可解群になることが分かる. 上の列で, G_0 はもちろん S_4, そして $G_1 = A_4$ も分かっている. G_2 は位数 4 の群でふたつの生成元を持って

いる. $[G_2, G_2] = \{1\}$ であるから, G_2 はアーベル群である. G_2 はクラインの 4 元群に同型なのではなかろうか. 実際,

```
gap> D[3];
Group([ (1,2)(3,4), (1,4)(2,3) ])
gap> IsElementaryAbelian(D[3]);
true
```

次に, $s8$ の導来列を調べてみると,

```
gap> DerivedSeries(s8);
[ Group([ (1,2), (1,2,3,4,5,6,7,8) ]),
  Group([ (1,2,3), (2,3,4), (2,4)(3,5), (2,6,4), (2,4)(5,7), (2,8,6,4)(3,5)]) ]
gap> List(last, Size);
[ 40320, 20160 ]
```

これは, $G = S_8$ の導来列が,

$$G = G_0 \supset G_1 = G_2 = \cdots$$

となることを示している. 言い替えれば, $G_0 = S_8, G_1 = A_8$ でその後は, $[A_8, A_8] = A_8$ となるので, $G_1 = G_2 = \cdots$ とどこまで行っても単位群には到達できないのである. したがって, S_8 も, A_8 も可解群ではない. GAP には, 直接与えられた群が可解群かどうかを判定する関数 `IsSolvableGroup( )` がある.

```
gap> IsSolvableGroup(s4);
true
gap> IsSolvableGroup(s8);
false
```

実は, n が 5 以上の時, S_n は可解群にはならないことが知られている. それが, 5 次以上の代数方程式に根の公式が存在しない (巾根と 4 則演算によって解けない) というアーベル, ガロアの理論の根拠になっている. より正確に言えば, 交代群 A_n は n が 5 以上のとき, 単純群になる. G が**単純群**とは, 自分自身と単位群以外に正規部分群を持たない群のことをいう. G が非可換な単純群だとすると, $[G, G]$ は正規部分群なので, $[G, G] = G$ または $[G, G] = \{1\}$ になる. しかし $[G, G] = \{1\}$ とすると, G はアーベル群になる. したがって $G = [G, G]$ となり, G は可解群にはならない. 実際, いろいろな意味で単純群は可解群のちょうど対極にある概念と考えることができる. 可解群はいくつかのアーベル群に分解できると考えられるが, 単純群は, それ以上は分解できない素粒子のようなものである. 有限群の構造を調べる問題は多くの場合, 単純群に帰着する. GAP は与えられた群が単純群かどうかを瞬時にして判定する.

```
gap> IsSimple(a8);
true
gap> IsSimple(s8);
false
gap> IsomorphismTypeInfoFiniteSimpleGroup(a8);
rec( series := "A", parameter := 8,
name := "A(8) ~ A(3,2) = L(4,2) ~ D(3,2) = O+(6,2)" )
```

当然のことながら, 有限単純群には皆名前がついている. `IsomorphismTypeInfoFiniteSimpleGroup(a8)` は有限単純群 `a8` についている名前をリストアップする関数である. 最初の名前 $A(8)$ は当然交代群 A_8 を意味する. ここでは, 交代群 A_n からできる有限単純群の無限系列を $''A''$ で表している. $A(8)$ 以降は, A_8 が他のリー型の有限単純群 $L(4,2)$, $O^+(6,2)$ と同型であることを示している.

(注意) これまでの議論では, 対称群 S_n を生成元を使って構成し, 交代群 A_n をその導来群として定義した. これは, 何事も不精をせずにちゃんと自分で作ってみなさい, という教育的配慮によるものであった. しかし, この辺で種明かしをしておくと, 実は GAP には, S_n も A_n も既に組み込まれている. これらを呼び出すのには次のようにすれば良い.

```
gap> s50 := SymmetricGroup(50);
Sym( [ 1 .. 50 ] )
gap> Size(s50);
30414093201713378043612608166064768844377641568960512000000000000
gap> a8 := AlternatingGroup(8);
Alt( [ 1 .. 8 ] )
gap> Size(a8);
20160
```

5.4 剰余類

G を群, H をその部分群とするとき, 集合 $Hg = \{hg \mid h \in H\}$ を H の**右剰余類** (right coset) という. G に $g \sim g' \Leftrightarrow g'g^{-1} \in H$ により同値関係が定義される. このとき, $g \in G$ を含む同値類が右剰余類 Hg に一致する. 同様に**左剰余類** gH も定義できる. 左剰余類 gH は, 同値関係 $g \sim g' \Leftrightarrow g^{-1}g' \in H$ に関する g を含む同値類である. G の右剰余類の集合を $H\backslash G$, 左剰余類の集合を G/H と, それぞれ表す. 各右剰余類から代表元をひとつずつ選んでできる G の部分集合を $H\backslash G$ の完全代表系という. $\{g_i \mid i \in I\}$ をひとつの完全代表系とすると (I は添字の集合),

$$G = \coprod_{i \in I} Hg_i$$

と互いに共通部分のない右剰余類の和集合として G が表される. G が有限群の場合, $|Hg| = |H|$ であるから, $|G| = |H||I|$ と書くことができる. この場合, $|I|$ は右剰余類 $H\backslash G$ の個数である. 特に部分群 H の位数 $|H|$ は G の位数 $|G|$ の約数であることが分かる (Fermat の小定理). $H\backslash G$ の元の個数を G の H による**指数** (index) といい, $[G : H]$ と表す. 以上の結果は左剰余類に関しても同様に成り立つ.

一般に, 右剰余類 Hg と左剰余類 gH は一致しない. 全ての $g \in G$ に対して $gH = Hg$ が成立することと, H が G の正規部分群になることは同値である. しかし, 次のような意味で右剰余類と左剰余類はつながっている.

$$(gH)^{-1} = H^{-1}g^{-1} = Hg^{-1}.$$

したがって, 右剰余類を調べることと, 左剰余類を調べることは本質的に同じであることに注意する.

GAP では G の部分群 H に対して `RightCosets(G, H)` により右剰余類の集合 $H\backslash G$ を求めることができる. また指数 $[G : H]$ は `Index(G, H)` により得られる. 例えば, $G = S_4$ とその部分群 $H =$`elab` に対し,

```
gap> s4:= Group((1,2), (1,2,3,4));;
gap> Index(s4,elab);
6
gap> RightCosets(s4,elab);
[ RightCoset(Group( [ (1,2), (3,4) ] ),()),
  RightCoset(Group( [ (1,2), (3,4) ] ),(2,3)),
  RightCoset(Group( [ (1,2), (3,4) ] ),(2,4,3)),
  RightCoset(Group( [ (1,2), (3,4) ] ),(1,2,3)),
  RightCoset(Group( [ (1,2), (3,4) ] ),(1,2,4,3)),
  RightCoset(Group( [ (1,2), (3,4) ] ),(1,3)(2,4)) ]
```

6 個の右剰余類 $\{Hg_1, \ldots, Hg_6\}$ の代表元が

$$g_1 = 1, \qquad g_2 = (2,3), \qquad g_3 = (2,4,3),$$
$$g_4 = (1,2,3), \qquad g_5 = (1,2,4,3), \qquad g_6 = (1,3)(2,4)$$

として得られた. 各行の `Group([ (1,2), (3,4)])` は `elab` を表す. しかし, これではいかにも見にくい. この群はそもそも `elab` として定義したのだから, きちんと名前を書いてくれても良さそうなものではないか. GAP とは言え, 何と融通の効かないことよと, ぼやきのひとつも出るが, これには理由がある. 実は `elab` は名前ではなく住所なのである. コンピュータのメモリー内に割り当てられた住所を指定しているのが `elab` である (これを identifier と呼ぶ). そこで, 群 `elab` に改めて名前をつけてやれば何かと便利であろう. GAP には名前をつける関数 `SetName` がある.

```
gap> SetName(elab, "2^2");
gap> elab;
2^2
```

どんな名前でも良いが, ここでは散文的ながら `elab` が $(\mathbf{Z}/2\mathbf{Z})^2$ に同型な基本アーベル群ということで, `2^2` と名付けることにする. もう一度, `RightCosets` を実行すると, 今度は

```
gap> RightCosets(s4,elab);
[ RightCoset(2^2,()), RightCoset(2^2,(2,3)),
  RightCoset(2^2,(2,4,3)), RightCoset(2^2,(1,2,3)),
  RightCoset(2^2,(1,2,4,3)), RightCoset(2^2,(1,3)(2,4)) ]
```

とすっきりと表される. ここで注意することは, GAP に指令を出すときにはあくまで `elab` を使い, 名前 `2^2` は使えないということである. 同姓同名の人がいるから名前だけでは手紙が届かない. 異なった群に同じ名前を付けることは許されるのである. 一方, 同じ住所に何人かが住むことは当然起こり得る. 6 畳一間のアパートに 2 人がつましく暮らすこともある. 群 `elab` に, 場合に応じて H, H' など好みの記号を使うことが出来る所以である.

さて, 右剰余類の代表元を直接リストとして取り出す時には次の様にする.

```
gap> rcosets := RightCosets(s4,elab);;
gap> rrep := List(rcosets, c -> Representative(c));
[ (), (2,3), (2,4,3), (1,2,3), (1,2,4,3), (1,3)(2,4) ]
```

Representative(c) は同値類 c から代表元を取り出す関数である. 当然のことながらこれはおまかせ定食であって, 自分好みの代表元を注文するには別料金 (別途の工夫) がいる. とにかくこれで, 先程の $g_1, \ldots, g_6$ がリストとして得られた. 左剰余類 $\{g'_1H, \ldots, g'_6H\}$ の代表系 $\{g'_1, \ldots, g'_6\}$ は, $g'_i = g_i^{-1}$ とすれば良い. これは次のようにすれば直接得られる.

```
gap> lrep := List(rcosets,  c -> Representative(c)^-1);
[ (), (2,3), (2,3,4), (1,3,2), (1,3,4,2), (1,3)(2,4) ]
```

すなわち

$$g'_1 = 1, \qquad g'_2 = (2,3), \qquad g'_3 = (2,3,4),$$
$$g'_4 = (1,3,2), \qquad g'_5 = (1,3,4,2), \qquad g'_6 = (1,3)(2,4).$$

ひとつの右剰余類 Hg を作るには, RightCoset(H, g) と書く.

```
gap> R_1 := RightCoset(elab, (1,2));
RightCoset(2^2,(1,2))
gap> (1,2,3) in R_1;
false
```

$g = (1,2)$ と $H =$elab に対し, $R_1 = Hg$ が定まった. 元 $(1,2,3)$ は Hg に含まれない. G の剰余類への分解 $G = \coprod_{i \in I} Hg_i$ において, 与えられた元 $g \in G$ がどの剰余類 Hg_i に含まれるか GAP に答えさせたかったら, 次のような関数を作ればよい.

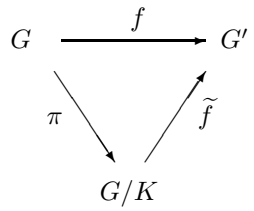
```
coset := function (G, H, g)
            local RC, xx;
            RC := RightCosets(G,H);
            for xx in RC do
                if g in xx then
                      return xx;
                fi;
            od;
          end;
```

関数 coset(G,H,g) により, g を含む剰余類が, RightCosets(G,H) で指定された代表元 $\{g_1, \ldots, g_k\}$ により $g \in Hg_i$ と与えられる.

```
gap> coset(s4, elab, (1,2,3,4));
RightCoset(2^2,(1,2,3))
gap> coset(s4,elab, (1,2,3,4)*(2,3,4));
RightCoset(2^2,(2,3))
gap> coset(s4,elab, (3,4)*(1,2,3));
RightCoset(2^2,(1,2,3))
```

5.5 商群, 準同型定理

G を群, H を G の正規部分群とする. このとき, G の右剰余類 Hg は左剰余類 gH に一致する. そこで, 以下 Hg を単に剰余類と呼ぶ. G の剰余類の集合 G/H に, $gH \cdot g'H = gg'H$ により積を定義することができ, それによって G/H に群の構造が入る. このようにして得られる群 G/H を G の H による**商群** (factor group) または**剰余群** (residue group) という. ここで, 群の準同型についても復習をしておく. G, G' を群とするとき, 写像 $f : G \to G'$ が**準同型写像** (homomorphism) とは任意の $g, g' \in G$ に対して $f(gg') = f(g)f(g')$ が成立することをいう. 今, H を G の正規部分群とするとき, $\pi : G \to G/H$ を $\pi(g) = gH$ と定めると, π は全射準同型になる. これを G から G/H への**自然な準同型写像** (natural homomorphism) という. 次に述べる準同型定理は全射準同型 $f : G \to G'$ が本質的に自然な準同型に一致することを主張している. $\mathrm{Ker}\, f = \{g \in G \mid f(g) = 1\}$ を準同型 f の**核** (kernel) という. $K = \mathrm{Ker}\, f$ は G の正規部分群になる. このとき, $f : G \to G'$ は次の図式を可換にするような同型写像 $\widetilde{f} : G/K \xrightarrow{\sim} G'$ を導く. これが準同型定理である.



これは, G' と G/K を $\widetilde{f}$ で同一視すると, 写像 f は写像 π と同じになることを意味する. 本来, 写像というのはこちらの世界にある G と彼方の世界に属する G' とを結びつける航路である. しかし全射準同型に限れば, その行き着く先は, G とその子分の K が幅を効かす身内のなわばりに過ぎない. 国際線に乗って新天地をめざすつもりが, 実は国内線だったというようなものである. 全射準同型は, あくまで G に関する情報しか教えてくれない. (でも知りたいのは G 自身だから ...). さて GAP では, 商群を `FactorGroup(G,H)` により定義する.

商群の具体例を扱うために, 少し準備をしよう. G の部分群 H に対して, H の G における**正規化群** (normalizer) を $N_G(H) = \{g \in G \mid g^{-1}Hg = H\}$ により定義する. $N_G(H)$ は, H を含む G の部分群で, その中で H が正規部分群となるようなもののうち最大のものである. 当然, $H \triangleleft N_G(H)$ である. GAP では, `Normalizer(G,H)` により $N_G(H)$ が計算できる. 先ず, A_8 の部分群 `elab3` を次のように定義する.

```
gap> elab3 := Group((1,2)(3,4)(5,6)(7,8), (1,3)(2,4)(5,7)(6,8),
                    (1,5)(2,6)(3,7)(4,8));;
gap> Size(elab3);
8
gap> IsElementaryAbelian(elab3);
true
gap> SetName(elab3, "2^3"); elab3;
2^3
```

elab3 は位数 8 の基本アーベル群になる. そこで, elab3 に 2^3 の名前を付ける. 次に, H =elab3 の正規化群 $N = N_G(H)$ を構成する.

```
gap> gap> norm:= Normalizer(a8, elab3);
Group([ (1,5)(2,6)(3,7)(4,8), (1,3)(2,4)(5,7)(6,8),
        (1,2)(3,4)(5,6)(7,8), (5,6)(7,8), (5,7)(6,8),
        (3,4)(7,8), (3,5)(4,6), (2,3)(6,7) ])
gap> Size(norm);
1344
gap> factor:= FactorGroup(norm, elab3);
Group([ (), (), (), (4,5)(6,7), (4,6)(5,7), (2,3)(6,7), (2,4)(3,5),
        (1,2)(5,6) ])
gap> Size(factor);
168
```

位数 1344 の群として norm $= N$ が定まり, その商群 factor $= N/H$ が位数 168 の群として得られた. N/H の生成元は置換として表されているが, これはあくまで剰余類の代表元であることに注意する.

ここで自然な準同型 $\pi : N \to N/H$ を考えよう. 一般に準同型 $f : G \to G'$ を GAP に認識させるのはそう簡単ではない. 写像を定義するには, 各 $g \in G$ に対して $f(g) \in G'$ を指定しなければならない. しかしこれは全然実用的ではない. もう少し利口なやり方は, G の各生成元の行き先を指定することである. もし f が準同型であることが分かっていれば, これで自動的に全ての G の元の行き先が決まる. この方法は後に試してみるが, 実際の場面では f が準同型かどうかは不明なことが多い. G の生成元 g に対して, $f(g) \in G'$ を指定したとき, それが準同型写像 f に拡張できるかどうかは, 一般に難しい問題である. しかし, 簡単に準同型が定義できる場合もあって, 自然な準同型 $\pi : G \to G/H$ もそのひとつである. GAP では,

```
gap> hom := NaturalHomomorphismByNormalSubgroup(norm, elab3);
<action epimorphism>
gap> ff := Image(hom);
Group([ (), (), (), (4,5)(6,7), (4,6)(5,7), (2,3)(6,7), (2,4)(3,5),
        (1,2)(5,6) ])
gap> ff = factor;
true
gap> Kernel(hom);
2^3
```

一行目が準同型 π =hom を定義する関数である. まさしく自然な準同型そのもので, 何の省略もなく公明正大, 実に分かりやすい命令であるが, これだけ長いと綴りを間違いなく書くのに苦労する. ズルをしてタブキーのお世話になるしか, 仕方がないところだろう. (By の B は大文字です. これはどうしてだろうと思案してみると, 要するに, 文章で命令を作る場合には, GAP はすべての単語を大文字で始めるという単純な原理にしたがっていることが分かる. すき間をあけられないのだから当然といえば当然であろうか.)

一旦準同型が定義されると, その像 (Image) と核 (Kernel) は, それぞれ `Image(hom)`, `Kernel(hom)` により, すぐさま計算できる. $\pi : N \to N/H$ は全射であるから $\pi(N) = N/H$, $\mathrm{Ker}(\pi) = K$ である.

写像 π による元 $x \in N$ の像 $\pi(x)$ および, $y \in N/H$ の逆像 $\pi^{-1}(y)$ は, それぞれ次のように表される.

```
gap> x := (1,8,3,5,7,6,2);; Image(hom, x);
(1,5,6,3,7,2,4)
gap> coset := PreImages(hom, last);
RightCoset(2^3,(2,8,6,7,3,4,5))
```

`Image(hom,x)` が像 $\pi(x)$ を与える. $y = \pi(x) \in N/H$ に対して `PreImages(hom, y)` が逆像 $\pi^{-1}(y)$ を与える. $\pi : N \to N/H$ であるから, $\pi(x_1) = y$ となる $x_1 = (2, 8, 6, 7, 3, 4, 5) \in N$ により $\pi^{-1}(y) = x_1 H$ (H による右剰余類) と表される. それが最後の式である. ここで注意すべきは, GAP は代表元を GAP 自身の規則によって選ぶので, x_1 が必ずしも最初に取った x と一致しないということである. しかし, もちろん x と x_1 は H の同じ剰余類に含まれるはずで、それは次のように確かめられる.

```
gap> rep := Representative( coset );
(2, 8, 6, 7, 3, 4, 5)
gap> x * rep^-1 in ker
 true
```

3 行目の命令が 「$xx_1^{-1} \in H$ が成立するや否や」と GAP に問うているわけで, それに対して, GAP が「然り」と答えているのである.

$f : G \to G'$ を全射準同型とすると, G の任意の部分群 H に対してその像 $f(H)$ は G' の部分群になる. 今, X を G の部分群の全体, Y を G' の部分群の全体とすると, 写像

$$\Phi' : X \to Y, \quad H \mapsto f(H)$$

が得られる. 一方 H' を G' の部分群とすると, H' の逆像 $f^{-1}(H')$ は G の部分群になる. $K = \mathrm{Ker}\, f = f^{-1}(\{1\})$ を f の核とすると, 当然 $f^{-1}(H') \supseteq K$ である. ここで, X の部分集合 $X_{\supseteq K}$ を K を含む G の部分群の全体として定義する. すると, 上のことから写像

$$\Psi : Y \to X_{\supseteq K}, \quad H' \mapsto f^{-1}(H')$$

が得られる. Φ' の $X_{\supseteq K}$ への制限 $X_{\supseteq K} \to Y$ を Φ とおく. すると, $\Psi(\Phi(H)) = H$, $\Phi(\Psi(H')) = H'$ ($H \in X_{\supseteq K}$, $H' \in Y$) が成り立つ. すなわち, Φ と Ψ は互いに他の逆写像となり, $\Phi : X_{\supseteq K} \to Y$ は全単射になることが分かる.

K を G の正規部分群として, 上の議論を自然な準同型 $\pi : G \to G/K$ に適用する. すると, G/K の部分群は全て, K を含む G の部分群 H により, $H' = \pi(H)$ として得られることが分かる. しかもこのような H は唯一つ定まる (実際 $H = \pi^{-1}(H')$). π の H への制限を π' とおくと, π' は H から H' への全射準同型で, $\mathrm{Ker}\, \pi' = K$. そこで準同型定理により, $H' \simeq H/K$ と表される. このようにして商群 G/K の部分群は全て G の K を含む部分群 H により, H/K の形で得られることが示される.

以上の議論を GAP で実地に確かめて見よう. 数学者はおのれの頭脳を何よりも信頼するという弱点 (?) を持つが, 我々はだまされやすい議論にたよらず, とにかく目で見て確認することにしよう. 下手な

考えと百聞は一見に然ずである．$f : G \to G'$ を準同型とする．H を G の部分群とするとき, GAP は `Image(f, H)` により, G' の部分群 $f(H)$ を定める．また, G' の部分群 H' に対して, `PreImages(f, H')` により逆像 $f^{-1}(H')$ を計算する．

先程の例 $\pi : N \to N/H$ に戻る．先ず適当な N の部分群を作ろう．

```
gap> t6 := Group((3,4), (3,4,5,6,7,8));
Group([ (3,4), (3,4,5,6,7,8) ])
gap> h1 := Intersection(t6, norm);
Group([ (5,6)(7,8), (5,7)(6,8), (3,4)(7,8), (3,5)(4,6) ])
gap> Size(h1);
24
gap> Intersection(h1, elab3);
Group(())
```

`t6` は 6 文字 $\{3, 4, \ldots, 8\}$ に関する置換群 (したがって S_6 と同型) である．`t6` と `norm` の共通部分を `h1` とおく．H_1 =`h1` は位数 24 の N の部分群であり, $H_1 \cap H = \{1\}$ になっている．特に, $H_1 \in X$ ではあるが, $H_1 \notin X_{\supseteq H}$ である．

```
gap> m1 := Image(hom, h1);
Group([ (4,5)(6,7), (4,6)(5,7), (2,3)(6,7), (2,4)(3,5) ])
gap> Size(m1);
24
gap> h2 := PreImages(hom, m1);
Group([ (1,2)(3,4)(5,6)(7,8), (1,3)(2,4)(5,7)(6,8), (1,5)(2,6)(3,7)(4,8),
        (5,6)(7,8), (5,7)(6,8), (3,4)(7,8), (3,5)(4,6) ])
gap> Size(h2);
192
gap> Intersection(h2, elab3) = elab3;
true
```

$M_1 = \pi(H_1)$ により, N/H の部分群 M_1 =`m1` が定義できた．当然 $M_1 \in Y$ である．M_1 も位数 24 の部分群であることに注意する．これは $\pi|_{H_1} : H_1 \to M_1$ が同型写像になっていることを示している．$\pi^{-1}(M_1) = H_2$ とおく．H_2 =`h2` は H を含む位数 192 の部分群である．したがって $H_2 \in X_{\supseteq H}$ である．さらに計算を続けると,

```
gap> m2 := Image(hom, h2);
Group([ (), (), (), (4,5)(6,7), (4,6)(5,7), (2,3)(6,7), (2,4)(3,5) ])
gap> m2 = m1;
true
```

$M_2 = \pi(H_2)$ とおく．すると, $M_1 = M_2$ が成り立つ．これは, H を含む N の部分群 H_2 と, N/H の部分群 M_1 に対して,

$$\pi(H_2) = M_1, \quad \pi^{-1}(M_1) = H_2$$

が成り立つことを示している．それが全単射 $X_{\supseteq H} \simeq Y$ の内容であった．

ここで商群 N/H の構造について調べておこう. 実は factor$= N/H$ は単純群になる.

```
gap> IsSimple(factor);
true
gap> IsomorphismTypeInfoFiniteSimpleGroup(factor);
rec( series := "L", parameter := [ 2, 7 ],
     name := "A(1,7) = L(2,7) ~ B(1,7) = O(3,7)
            ~ C(1,7) = S(2,7) ~ 2A(1,7) = U(2,7) ~ A(2,2) = L(3,2)" )
```

factor は位数 168 の単純群である. 5.3 と同様に単純群についての情報を求めてみる. "L" は factor が Lie 型の単純群であることを示している. parameter : [2,7] は factor が $L(2,7)$ で表される単純群, すなわち $SL_2(\mathbb{F}_7)/\{\pm 1\}$ (7 元体 $\mathbb{F}_7$ 上の, 行列式 1 の 2 次行列全体のなす群 $SL_2(\mathbb{F}_7)$ をその中心 $\{\pm 1\}$ で割った商群) に同型になることを示している. またそれは同時に, $L(3,2)$ すなわち $SL_3(\mathbb{F}_2)$ (2 元体 $\mathbb{F}_2$ 上の行列式 1 の 3 次行列全体のなす群) にも同型になる.

5.6 群の作用

群というものについて, つらつら思いをめぐらすと, 群が単独でその辺に転がっているわけではないことに気が付くだろう. 歴史的にも, 群は代数方程式の根の置換を記述するものとして登場した. 方程式があってこその群である. また正多面体群, 結晶群, 運動群と思い付くままに並べてみても, それぞれ群の活躍する舞台, すなわち, 正多面体, 結晶, 運動が起こっている空間, といったものが常に存在するのである. 群それ自身を取り上げて抽象的に研究する立場はもちろんあるが, やはり, 群が輝きを増し, その特異な能力を遺憾なく発揮するのは, 群が何かあるものに関わったときである. 哲学的かつ宗教的に, 演歌的風味を添えて表現すると, 「他者に働きかけることによってこそ群の生きる道がある」ということだろうか.

この群が他者に働きかけるということが, 取りもなおさず群の作用である. 数学的な定義を復習しておこう. 今, 群 G が集合 X に (右から) **作用** するとは, 次の条件 (i), (ii) を満たす写像 $X \times G \to X, (x,g) \mapsto xg$ が存在することを言う.

(i) 任意の $x \in X$ に対して $xe = e$ (e は G の単位元),
(ii) 任意の $x \in X, g, h \in G$ に対して $(xg)h = x(gh)$.

さて今, G が X に作用しているとする. $g \in G$ に対し, $\varphi_g : X \to X, x \mapsto xg$ により X から X への写像 φ_g が定まる. $\varphi_g \circ \varphi_h = \varphi_{gh}$, $\varphi_e = \mathrm{id}_X$ が成り立ち, φ_g は X から X への全単射になる. X から X への全単射の全体の集合を $\mathrm{Aut}(X)$ と表す. 写像の結合により, $\mathrm{Aut}(X)$ は群になる. 先に注意したことから, $g \mapsto \varphi_g$ により, 準同型 $\varphi : G \to \mathrm{Aut}(X)$ が定まる. φ を**作用準同型** (Action homomorphsim) という. X が n 個の元からなる集合の場合, $\mathrm{Aut}(X) \simeq S_n$ であり, 準同型 $\varphi : G \to S_n$ が得られる. 逆に, 準同型 $\varphi : G \to \mathrm{Aut}(X)$ が与えられると, $(x,g) \mapsto (x)\varphi_g$ により, G の X への作用が構成される. そこで, G の X への作用を与えることと, 準同型 $G \to \mathrm{Aut}(X)$ を与えることとは, 同値になる.

ここで群の作用の基本的な例をいくつかあげておこう. まず, $X = G$ として, $X \times G \to X$ を $(x,g) \mapsto xg$ で定義すると, G の G への作用が得られる. これを G の**右移動** (right translation) という. 同様に $X \times G \to X$ を $(x,g) \mapsto g^{-1}x$ も G の G への 作用になる. これを**左移動** (left translation) という. この 2 つの作用は群の積演算を単に反映しているに過ぎない. しかし, 同じ $X = G$ でも $(x,g) \mapsto g^{-1}xg$ としてやると別の作用が得られる. これを G の (G 自身への) **共役** (conjugate) の作用という. しかし

X の取り方としては G だけではなく, 多くの選択枝があり, それにより群の世界が一挙に広がるのである. 対称群 $G = S_n$ の場合, $X = \{1, 2, \ldots, n\}$ とすると, G は X 上に n 文字の置換として作用する, すなわち

$$\sigma = \begin{pmatrix} 1 & 2 & \cdots & n \\ i_1 & i_2 & \cdots & i_n \end{pmatrix} \in S_n$$

に対し, $X \times G \to X, (x, \sigma) \mapsto x\sigma = i_x$. この置換の作用は, あまりにも基本的で S_n の定義そのものである. これだけを考える分には, わざわざ, しかめっつらをして「群の作用」などといばる必要もない. しかしこの一見素朴な, 小学生でも分かる作用 (チンパンジーのアイちゃんは単語を並べ換えて文章を作るから, 置換をちゃんと理解していると言えるだろう) が意外と役に立つのである. 例えば, $Y = X \times X = \{(i, j) \mid 1 \leq i, j \leq n\}$ とすると $G \subset S_n$ は, $Y \times G \to Y, ((i, j), \sigma) \mapsto (i\sigma, j\sigma)$ により, Y に作用する. これを少し変形して, Z を 2 個の元からなる X の部分集合の全体としても, やはり G は Z に作用する. あるいは, 2 個を k 個に置き換えても作用が得られる. このようにして, どんどん新しい作用が得られる. この辺になって来ると見掛け上は込み入って見えるので, 高校生でもちょっと怪しくなる. 2 個でも k 個でも同じことさ, とすましていられるのは数学者だけかも知れない. チンパンジーが直積集合 (2 個の場合) を理解できるかどうかは興味のある所である.

さて GAP では, 上に述べた G の G 自身への作用, $G \subset S_n$ の $X = \{1, 2, \ldots, n\}$ への置換群としての作用, およびそれから導かれる作用が基本になる. 既に第 1 章で説明したように, $G \subset S_n$ の点集合 $X = \{1, 2, \ldots, n\}$ への作用は, 普通の演算と同様にして計算できる.

```
gap> g := (1,2)*(2,3,4);;
gap> 3^g;
4
```

文字 3 が置換 g によって $3g = 4$ に移された. これをもう少し点集合への作用を前面に出して書くと,

```
gap> OnPoints(3,g);
4
```

となる. しかし `OnPoints` は便利な記号で, G の共役の作用にも使われる. すなわち

```
gap> OnPoints((1,2,3),g);
(1,4,3)
gap> g^-1*(1,2,3)*g;
(1,4,3)
```

これは, `OnPoints((1,2,3),g)` が $g^{-1}(1, 2, 3)g$ に一致することを示している. さらに, G への右移動および左移動については,

```
gap> OnRight((),g);
(1,3,4,2)
gap> OnLeftInverse((),g);
(1,2,4,3)
```

単位元 e に右移動により g を作用させれば $eg = g = (1,3,4,2)$ が得られる. 左移動については `OnLeft` ではなく, g^{-1} を左からかけるという意味で, `OnLeftInverse` となることに注意する. 直積集合 Y (すなわち数の組の集合) と 2個の元からなる X の部分集合の集合 Z への作用は, それぞれ

```
gap> OnPairs([2,3],g);
[ 1, 4 ]
gap> OnSets([2,3],g);
[ 1, 4 ]
```

となる. ここでは, Y で考えても, Z で考えても差はない. 後に違いの分かる例をあげる. 2 個の数の組ではなく, もっと多い組については, `OnTuples` を使う.

```
gap> OnTuples([1,2,4,5],g);
[ 3, 1, 2, 5 ]
```

これは, $3 = 1g, 1 = 2g, 2 = 4g, 5 = 5g$ を意味する.

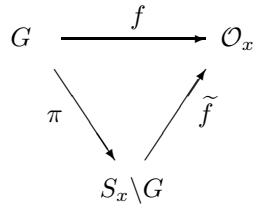
G が集合 X に作用しているとする. X の元 x, y に対し, $y = xg$ となる $g \in G$ が存在するとき, $x \sim y$ と表す. すると, 関係 $x \sim y$ は X に同値関係を定める. これは, G が X に作用することの帰結である. この関係 $\sim$ に関する同値類 $\mathcal{O}$ は $x \in \mathcal{O}$ を選ぶと, $\mathcal{O} = \{xg \mid g \in G\}$ と表される. $\mathcal{O} = \mathcal{O}_x$ を x を通る G の**軌道** (orbit) という. $\sim$ が同値類であることから, X は共通部分を持たないいくつかの軌道の和集合 (disjoint union) に分解される.

$$X = \coprod_{x \in X/G} \mathcal{O}_x.$$

ここに, X/G は同値類の集合 (あるいは, その完全代表系) を表す. 群の作用において, 軌道のイメージは決定的に重要である. X の一点 x を G の作用によってどんどん移して行く. 例えば, 加法群 $G = \mathbb{R}$ がユークリッド空間 $\mathbb{R}^3$ に作用しているとする. G の作用につれて, $\mathbb{R}^3$ 内に点 x を通る軌跡が描かれていく. これが x を通る軌道である. x を含む軌道と言ってもいいのだが, ここはやはり, **通る** という語感を大事にしたい. ふたつの異なった軌道は, お互いにごく近くを通ることはあっても, 決して交わらない. 隣りを走る汽車の窓に一瞬の面影を認めたとしても, 運命が交差することはないのである.

$x \in X$ に対し, $St_G(x)$ を点 x を動かさない G の元の全体とする, すなわち, $St_G(x) = \{g \in G \mid xg = x\}$. $St_G(x)$ は G の部分群になる. これを G における x の**固定化群** (stabilizer) という. G-軌道 $\mathcal{O}_x$ と固定化群 $St_G(x)$ とは以下述べるように, さんまと大根おろしのような, 切っても切れない関係にある. $St_G(x) = S_x$ と書く. 今, 写像 $f : G \to \mathcal{O}_x$ を $f(g) = xg$ により定義する. ($\mathcal{O}_x \subset X$ は群ではないから, f は準同型ではない. 念のため). f は全射になる. ここで, $\pi : G \to S_x\backslash G$, $g \mapsto S_x g$ を自然な写像とする. ただし, $S_x\backslash G$ は G の群 S_x による右剰余類の集合である. S_x は必ずしも正規部分群ではないので, $S_x\backslash G$ は群とは限らない. このとき, $f : G \to \mathcal{O}_x$ は, 次の図式を可換にするような全単射 $\widetilde{f} : S_x\backslash G \to \mathcal{O}_x$ を導く.

(5.6.1)



さらに, 全単射 $S_x\backslash G \simeq \mathcal{O}_x$ のもとに, $\mathcal{O}_x$ への G の作用は $S_x\backslash G$ への自然な G の作用 (G の右移動) に一致する. 読者は, この結果と準同型定理との類似が気になるだろう. 実際, 準同型定理はこの定理の特別な場合に過ぎない. 今, $f : G \to G'$ を全射準同型とする. $X = G'$ とし, X への G の作用を, $X \times G \to X, (x, g) \mapsto xf(g)$ (G' での積) で定義する. $x = e$ を G' の単位元とすると, $\mathcal{O}_x = X = G'$, $St_G(x) = \mathrm{Ker}\, f$ であることがすぐ分かる. このとき, 上の図式は準同型定理そのものを表す. この意味で上の結果が準同型定理の遥かなる拡張になっていることに気がつくだろう. ところで, 準同型定理の項 (5.5) で全射準同型は, 新世界へのパスポートにはならないことに注意した. G-軌道 $\mathcal{O}_x$ でも似たような事情が起きる. G-軌道として, いろいろ目新しいものが取れそうにも思えるが, 実際にはそこへの G の作用は, G のある部分群による剰余類の集合への G の作用と一致してしまう. 我々は, やはりお釈迦様の手の平で踊っているに過ぎない. 群 G の栄光は, このようにして G の支配する世界の隅々にまで及んでいるのである.

有限群 G が 有限集合 X に作用しているとする. このとき (5.6.1) より

(5.6.2) $$|\mathcal{O}_x| = |G|/|St_G(x)|$$

が成り立つ. 特に, 一つの軌道に含まれる元の個数は常に G の位数の約数になる, という著しい結果が得られる.

GAP では, $x \in X$ の G-軌道を次のようにして計算する. 置換群 G の点集合 $X = \{1, 2, \ldots, n\}$ への作用から誘導される作用, `OnPoints`, `OnPairs`, `OnSets` など, によって与えられている場合には,

```
gap> a4:= Group((1,3,2), (2,4,3));
Group([ (1,3,2), (2,4,3) ])
gap> Orbit(a4,1, OnPoints);
[ 1, 3, 2, 4 ]
gap> Orbit(a4,1);
[ 1, 3, 2, 4 ]
gap> Orbit(a4, [1,2], OnPairs);
[ [ 1, 2 ], [ 3, 4 ], [ 2, 1 ], [ 4, 3 ], [ 1, 3 ], [ 4, 2 ],
  [ 3, 1 ], [ 2, 4 ], [ 1, 4 ], [ 2, 3 ], [ 4, 1 ], [ 3, 2 ] ]
gap> Orbit(a4, [1,2], OnSets);
[ [ 1, 2 ], [ 3, 4 ], [ 1, 3 ], [ 2, 4 ], [ 1, 4 ], [ 2, 3 ] ]
```

として計算できる. 最初の例では, $G = A_4$ の点 $1 \in X$ の軌道 $\mathcal{O}_1 = \{1^g \mid g \in G\}$ を計算している. この場合, 点集合 X への作用であるから, `Orbit(a4, 1, OnPoints)` と最後に `OnPoints` を加えるが, これは省略しても構わない. 2 番目と 3 番目の例は, G の `[1,2]` を通る軌道である. この場合, `[1,2]` を Y の元と見るか, Z の元として見るかによって答えが変わる. Y の元として見る場合は直積集合への

作用であるから, OnPairs と指定する. 一方, Z への作用は, [1,2] を集合 $\{1,2\}$ として見た作用であるから OnSets と書く. 前に見たように, r 個の数の組を扱う場合には, OnTuples を使えばよい. さらに悪のりして, G の 組の組 (すなわち $Y \times Y$) への作用を調べるときは, OnTuplesTuples, (あるいは OnPairsPairs), 集合の集合 ($\{\{1,2\},\{3,4\}\}$ の形のもの全体, これは $Z \times Z$ とは異なる) への作用を調べるときは, OnSetsSets を使う. さらにややこし好きの人のためには, 組の集合への作用 OnTuplesSets ($\{[1,2],[3,4]\}$ の形のもの全体), 集合の組への作用 OnSetsTuples ($[\{1,2\},\{3,4\}]$ の形のもの全体) も考えられるが (ここで $\{1,2\}$ は集合, $[1,2]$ は組を表す) ここでは省略する. 各自試みられたい.

```
gap> Orbit(a4,[[1,2],[3,4]], OnTuplesTuples);
[ [ [ 1, 2 ], [ 3, 4 ] ], [ [ 3, 1 ], [ 2, 4 ] ], [ [ 1, 4 ], [ 2, 3 ] ],
  [ [ 2, 3 ], [ 1, 4 ] ], [ [ 2, 1 ], [ 4, 3 ] ], [ [ 3, 4 ], [ 1, 2 ] ],
  [ [ 1, 3 ], [ 4, 2 ] ], [ [ 4, 2 ], [ 1, 3 ] ], [ [ 4, 1 ], [ 3, 2 ] ],
  [ [ 2, 4 ], [ 3, 1 ] ], [ [ 3, 2 ], [ 4, 1 ] ], [ [ 4, 3 ], [ 2, 1 ] ] ]
gap> Orbit(a4,[[1,2],[3,4]], OnSetsSets);
[ [ [ 1, 2 ], [ 3, 4 ] ], [ [ 1, 3 ], [ 2, 4 ] ], [ [ 1, 4 ], [ 2, 3 ] ] ]
```

以上の議論では, 作用の仕方を指定した上で, 与えられた点を通る軌道を計算した. しかしもっと重要なのは, G が集合 X に作用している時に, X を G の軌道に分解することであろう. そこで先ず, X としてもう少し複雑なものを考えよう. GAP には配置 (arrangement) という関数が用意されている.

```
gap> Arrangements([1..3]);
[ [  ], [ 1 ], [ 1, 2 ], [ 1, 2, 3 ], [ 1, 3 ], [ 1, 3, 2 ], [ 2 ], [ 2, 1 ],
  [ 2, 1, 3 ], [ 2, 3 ], [ 2, 3, 1 ], [ 3 ], [ 3, 1 ], [ 3, 1, 2 ], [ 3, 2 ],
  [ 3, 2, 1 ] ]
```

この例から分かるように, Arrangements([1..3]) は $1 \sim 3$ の中から何個か選んで, 重複しないように並べた順列の集合である. 選ぶ個数を指定することもできる. 数 $1 \sim 4$ の中から 3 個の数を重複しないように選んでできる順列の集合は, Arrangements([1..4],3) で与えられる.

```
gap> omeg := Arrangements([1..4],3);
[ [ 1, 2, 3 ], [ 1, 2, 4 ], [ 1, 3, 2 ], [ 1, 3, 4 ], [ 1, 4, 2 ],
  [ 1, 4, 3 ], [ 2, 1, 3 ], [ 2, 1, 4 ], [ 2, 3, 1 ], [ 2, 3, 4 ],
  [ 2, 4, 1 ], [ 2, 4, 3 ], [ 3, 1, 2 ], [ 3, 1, 4 ], [ 3, 2, 1 ],
  [ 3, 2, 4 ], [ 3, 4, 1 ], [ 3, 4, 2 ], [ 4, 1, 2 ], [ 4, 1, 3 ],
  [ 4, 2, 1 ], [ 4, 2, 3 ], [ 4, 3, 1 ], [ 4, 3, 2 ] ]
```

対称群 S_4 は OnTuples の作用により 集合 Ω =omeg に作用する. 部分群 A_4 の Ω への作用による軌道分解は, 関数 OrbitsDomain により得られる.

```
gap> OrbitsDomain(a4, omeg, OnTuples);
[ [ [ 1, 2, 3 ], [ 3, 1, 2 ], [ 1, 4, 2 ], [ 2, 3, 1 ],
    [ 2, 1, 4 ], [ 3, 4, 1 ], [ 1, 3, 4 ], [ 4, 2, 1 ],
    [ 4, 1, 3 ], [ 2, 4, 3 ], [ 3, 2, 4 ], [ 4, 3, 2 ] ],
  [ [ 1, 2, 4 ], [ 3, 1, 4 ], [ 1, 4, 3 ], [ 2, 3, 4 ],
    [ 2, 1, 3 ], [ 3, 4, 2 ], [ 1, 3, 2 ], [ 4, 2, 3 ],
```

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

ちょっと見にくいが, Ω が2つの軌道に分解していることが分かる. さらに S_4 の部分群 elab を取ると,

```
gap> OrbitsDomain(elab, omeg, OnTuples);
[ [ [ 1, 2, 3 ], [ 2, 1, 3 ], [ 1, 2, 4 ], [ 2, 1, 4 ] ],
  [ [ 1, 3, 2 ], [ 2, 3, 1 ], [ 1, 4, 2 ], [ 2, 4, 1 ] ],
  [ [ 1, 3, 4 ], [ 2, 3, 4 ], [ 1, 4, 3 ], [ 2, 4, 3 ] ],
  [ [ 3, 1, 2 ], [ 3, 2, 1 ], [ 4, 1, 2 ], [ 4, 2, 1 ] ],
  [ [ 3, 1, 4 ], [ 3, 2, 4 ], [ 4, 1, 3 ], [ 4, 2, 3 ] ],
  [ [ 3, 4, 1 ], [ 3, 4, 2 ], [ 4, 3, 1 ], [ 4, 3, 2 ] ] ]
```

今度は, Ω が elab の 6 個の軌道に分解する. OrbitsDomain(G, X, act) の書式は, 最初に作用する群 G, 次いで作用域 X, 最後に作用の種類 act を書く. GAP には, このようにして分解した各軌道に含まれる元の個数を与える関数が用意されている.

```
gap> OrbitLengthsDomain(a4, omeg, OnTuples);
[ 12, 12 ]
gap> OrbitLengthsDomain(elab, omeg, OnTuples);
[ 4, 4, 4, 4, 4, 4 ]
```

一方, Domain を経由せずに, 直接 Ω 上の軌道分解やその長さを計算することも出来る.

```
gap> Orbits(a4, omeg, OnTuples);;
gap> OrbitLengths(a4, omeg, OnTuples);;
```

同じ結果が得られるが, しかし計算時間は Domain を使う方が速いことが多い.

今まで, Arrangements([1..4],3) により作用域を指定して来たが, 同様の命令に Combinations がある. こちらは順列ではなく, 組合せを与える. すなわち Combinations([1..5],3) は $1 \sim 5$ の中から 3 個の数を選んで出来る全ての集合を要素とする集合である.

```
gap> omeg1 := Combinations([1..5],3);
[ [ 1, 2, 3 ], [ 1, 2, 4 ], [ 1, 2, 5 ], [ 1, 3, 4 ], [ 1, 3, 5 ],
  [ 1, 4, 5 ], [ 2, 3, 4 ], [ 2, 3, 5 ], [ 2, 4, 5 ], [ 3, 4, 5 ] ]
```

omeg1 の上への a4 および, elab の作用を考えると,

```
gap> OrbitLengthsDomain(a4, omeg1, OnSets);
[ 4, 6 ]
gap> OrbitLengthsDomain(elab, omeg1, OnSets);
[ 2, 1, 2, 4, 1 ]
```

が得られる. omeg1 の元は集合であるから, 作用は OnTuples ではなく OnSets にしなければならないことに注意する.

G の X への作用が与えられたとき, G における $x \in X$ の 固定化群 $St_G(x)$ は次のように計算される.

```
gap> Stabilizer(a8, [1,2,3], OnTuples);
Group([ (4,5,6), (4,5,7), (5,8,6) ])
gap> Size(last);
60
gap> Stabilizer(a8, [1,2,3], OnSets);
Group([ (6,7,8), (5,6,7), (5,8,6), (4,5,6), (4,5,7), (1,2,3),
        (2,3)(5,7,6,8) ])
gap> Size(last);
360
```

最初の例は, 集合 $Z = \{1, 2, 3\}$ の各点を固定する $G = A_8$ の元の全体, 2 番目の例は, Z を集合として固定する元の全体である. 見ての通り, 後者のサイズは圧倒的に大きくなる. ここで、 (5.6.2) の等式を直接確かめておこう.

```
gap> OrbitLength(a8, [1,2,3], OnSets);
56
gap> OrbitLength(a8, [1,2,3], OnTuples);
336
gap> 56*360;
20160
gap> 336*60;
20160
```

確かに, $|St_G(x)||\mathcal{O}_x| = |G|$ が成立している.

本節の冒頭に述べたように, G の X への作用を与えることと, 準同型 $G \to \mathrm{Aut}(X) = S_n$ ($n = |X|$) を与えることは同値である. GAP では, `Actionhomomorphsim` により作用から準同型が得られる.

```
gap> hom := ActionHomomorphism(a4, omeg, OnTuples);
<action homomorphism>
gap> Image(hom);
Group([(1,9,13)(2,10,14)(3,7,15)(4,8,16)(5,12,17)(6,11,18)(19,22,23)(20,21,24),
       (1,4,5)(2,3,6)(7,14,19)(8,13,20)(9,17,21)(10,18,22)(11,15,23)(12,16,24) ])
gap> Size(Range(hom));
620448401733239439360000
gap> Size(Image(hom));
12
```

`hom` は準同型 $f : G \to S_{24}$ を与える. 当然の事ながら, `hom` は全射ではない. `hom` の値域は S_{24} であり, 上の巨大な数は, $620448401733239439360000 = 24!$ を表す. hom の値域を $f(G) \subset S_{24}$ に制限したければ, `"surjective"` を付け加えれば良い.

```
gap> hom1 := ActionHomomorphism(a4, omeg, OnTuples, "surjective");
<action epimorphism>
gap> Size(Range(hom1));
12
```

これより, hom1 として全射準同型 $G \to f(G)$ が得られる.

5.7 群自身への群の作用 – p-シロー群への応用 –

前節では, G の点集合への置換による作用を考えた. 一方, G を G 自身に作用させることもできる. これはなにやら, 自分自身を尻尾から食べていって最後に消えてしまう蛇を連想してしまう (ジョジョのお話を知っている人はバニラアイスを思い出して下さい). 群はいくら作用させても減るものではないからこのような破滅的な事態に陥ることはないが, 自家中毒のような印象も受ける. しかし, この作用が意外に有効で, 強力な結果をもたらすのである. 例えば, 有限群 G を右移動によって G に作用させると, G は $n = |G|$ 個の元の置換を引き起こし, これによって G を S_n の部分群とみなすことができる. また, G の種々の部分群の集合の上への, G の部分群 H の作用を考えることにより実に多くの作用が生まれ, 思いがけない応用が得られる.

一般に, G が X に作用していれば X は G-軌道の disjoint union に分解する. 各 G 軌道への作用は別物と考えられるので, X が唯一つの軌道からなっているときが, 本質的に重要である. このとき, G は X に**推移的** (transitive) に作用しているという. これは, X の点 x, y に対して $y = xg$ となる元 $g \in G$ が必ず存在することと同値である. 今, G が X に推移的に作用しているとする. すると $x \in X$ を取れば, $X = \mathcal{O}_x$ となる. $H = St_G(x)$ を x の固定化群とすると, (5.6.1) より G の X への作用は, 右剰余類の集合 $H\backslash G$ への G の右移動による作用に一致する. その意味で, 推移的な作用はすべて群自身への群の作用に他ならない.

ここで, 群自身への作用の典型的な応用例を見るために, p-シロー部分群を導入する. G を有限群とする. 素数 p に対して, G の位数を割る最大の p 巾を p^m をとする. G には位数 p^m の部分群が存在し, それらはすべて G で共役になる. このような部分群を G の **p-シロー部分群** (p-Sylow subgroup) という. より一般に位数が p の巾である群を **p 群** (p-group) という. p-シロー群は G に含まれる最大位数の p-部分群である. p-シロー群の存在については後に触れることにして, ここでは, 存在するものとして議論する. 今 H を G の p-部分群とする. このとき G には H を含む p-シロー群が必ず存在することを示そう. G の p-シロー群 P をひとつ取り,

$$\Omega = \{g^{-1}Pg \mid g \in G\}$$

とおく. G は共役の作用により Ω に作用し, 当然のことながらこの作用は推移的になる. そこで $|\Omega| = |G|/|St_G(P)|$ が成り立つ. ここで $St_G(P) = \{g \in G \mid g^{-1}Pg = P\}$ は P を含むので, $St_G(P)$ の位数は p^m の倍数, したがって, $|\Omega|$ は p と互いに素になる. さて G の部分群 H も Ω に作用する. その軌道分解を

$$\Omega = \coprod_{x \in \Omega/H} \mathcal{O}_x \tag{5.7.1}$$

とする. H は p-群であるから, $|\mathcal{O}_x| = |H|/|St_H(x)|$ は p の巾になる. すなわち, 各 $\mathcal{O}_x$ に対して, $|\mathcal{O}_x|$ は p で割り切れるかまたは, $|\mathcal{O}_x| = p^0 = 1$. ここで $|\Omega|$ は p と素であったことを思い出すと, Ω には必ず $|\Omega_x| = 1$ となる H-軌道の存在することが分かる. そのような軌道 $\mathcal{O}_x$ を取り, $\mathcal{O}_x = \{P_1\}$ とおく. P_1 は P に共役な p-シロー群である. $H \subseteq P_1$ となることを示そう. $\mathcal{O}_x = \{P_1\}$ より, $H \subseteq N_G(P_1)$. このとき (第２) 準同型定理により, HP_1 は P_1 を含む位数 p 巾の部分群になることが分かる. P_1 は p-シロー群であるから, これは $H \subseteq P_1$ を意味する. 以上で, H を含む p-シロー群の存在が示された. (ここ

で特に H として G の任意の p-シロー群を取ると, $H = P_1$ となり, H は P に G-共役になることも導かれる.) Ω の H-軌道への分解をよく見てみると, H を含む p-シロー群の個数は $|\mathcal{O}_x| = 1$ となる H-軌道の個数に等しいことが分かる. 実際, $\mathcal{O}_x = \{P_2\}$ ならば P_2 は H を含み, 逆に シロー群 $P_2 \in \Omega$ が H を含めば, 明らかに P_2 を通る H-軌道は $\{P_2\}$ である.

ここで $H = P$ の場合を考える. $|\mathcal{O}_x| = 1$ となる H-軌道は $\mathcal{O}_x = \{P\}$ のみであり, その他の $\mathcal{O}_y$ については $|\mathcal{O}_y|$ はすべて p で割り切れる. これより $|\Omega| \equiv 1 \pmod{p}$. Ω は G のすべての p-シロー群の集合であるから, G の p-シロー群の個数は $\mod p$ で 1 になることが分かる. 以上の議論から次が得られる. ((i) については後に議論する).

(5.7.2)

(i) G には p-シロー部分群が存在する.
(ii) G の p-シロー群はすべて G-共役になり, $\sharp\{$ G の p-シロー群 $\} \equiv 1 \pmod{p}$ が成り立つ.
(iii) H を G の p-部分群とすると, H を含む p-シロー群が存在する. その個数は, (5.7.1) の軌道分解における長さ 1 の軌道の個数で与えられる.

上の議論から見て取れるように, この証明の本質は G の Ω への作用である. そこで, GAP を利用して上のプロセスを実行し, H を含む p-シロー群 P_1 を具体的に求めてみよう. GAP にはシロー部分群を計算する関数が用意されている. 後に, p-シロー群を実際に構成してみるが, ここでは安直に GAP の関数を利用することにする. GAP では, 群 G の p-シロー群は `SylowSubgroup(G,p)` で与えられる. 数ある p-シロー群のうちの一つが答えとして帰って来るのである. どの群が戻って来るかは, 神 (GAP) ならぬ我々には知る由もない. さて G として S_5 を取り, その 2-シロー群 ($p = 2$ の場合) を考える.

```
gap> s5 := Group((1,2), (1,2,3,4,5));;
gap> u:= SylowSubgroup(s5,2);
Group([ (4,5), (2,3), (2,4)(3,5) ])
gap> Size(u);
8
```

G は位数 $24 = 2^3 3$ であるから, $P =$`u` は確かに G の 2-シロー部分群になっている. G の部分群 H として位数 4 の部分群 `elab` を取ろう.

```
gap> IsSubgroup(u, elab);
false
```

群 $H =$ `elab` は 2-シロー群 P には含まれていない. そこで H を含む 2-シロー群 P_1 を見つけることにする. 先程の議論のように, $\Omega = \{g^{-1}Pg \mid g \in G\}$ への G の共役の作用を考える. GAP では

```
gap> omega := ConjugacyClassSubgroups(g,u);
Group( [ (4,5), (2,3), (2,4)(3,5) ] )^G
```

のように書けば, `omega` $= \Omega$ とその上への G の共役による作用が確定する. 5.6 節のように, `omega` を `elab` の軌道に分解し, 各軌道に含まれる元の個数をリストアップしてみる.

```
gap> OrbitLengthsDomain(elab, omega);
[ 4, 2, 2, 1, 2, 4 ]
```

これは, omega が 6 個の軌道に分解し, 4 番目の軌道は唯ひとつの元からなることを示している. 前の議論では, この 4 番目の軌道に含まれる元 P_1 が H を含む 2-シロー群になるのであった. そこで, 取りあえず各軌道の代表元を求めよう. OrbitsDomain を定義し, 5.4 節の剰余類の代表元の場合と同様の命令を使う.

```
gap> orbits := OrbitsDomain(elab, omega);;
gap> rep := List(orbits, c -> Representative(c));
[ Group([ (3,4), (2,5), (2,3)(4,5) ]), Group([ (4,5), (2,3), (2,4)(3,5) ]),
  Group([ (1,2), (4,5), (1,4)(2,5) ]), Group([ (3,4), (1,2), (1,3)(2,4) ]),
  Group([ (2,3), (1,4), (1,2)(3,4) ]), Group([ (2,3), (1,5), (1,3)(2,5) ]) ]
```

4 番目の代表元を P_1 =u1 とおく. P_1 は H を含む 2-シロー群であることが以下のように確かめられる.

```
gap> u1 := rep[4];
Group([ (3,4), (1,2), (1,3)(2,4) ])
gap> Size(u1);
8
gap> IsSubgroup(u1,elab);
true
```

一つの元からなる軌道は 4 番目の $\{P_1\}$ のみなので, 前の議論によると P_1 が H を含む唯一の 2-シロー群になるはずである. これを各軌道の代表元を使って直接確かめてみよう.

```
gap> gap> kk := [];;
gap> for i in [1..6] do
>     if IsSubgroup(rep[i], elab) = true then
>         kk[i] := 1;
>     else
>         kk[i] := 0;
>     fi;
> od;
gap> kk;
[ 0, 0, 0, 1, 0, 0 ]
```

各軌道の代表元 rep[i] が H =elab を含めば 1, 含まなければ 0 を与えるようにする. 確かに 4 番目のみが 1 を返していることが見てとれるだろう.

次にもう少し大きな例を考える. P =syl2 を G =a8 の 2-シロー群とする.

```
gap> syl2 := SylowSubgroup(a8,2);
Group([ (1,8)(5,7), (2,6)(5,7), (3,4)(5,7), (3,5)(4,7),
  (1,2)(6,8),  (1,3)(2,5)(4,8)(6,7) ])
gap> Size(syl2);
64
```

また, H =elab3 を 5.3 節に定義した a8 の位数 8 の部分群とする. H は P に含まれない.

```
gap> IsSubgroup(syl2, elab3);
false
```

そこで, H を含む G の 2-シロー群をすべて求めてみよう. 例によって $\Omega = \{g^{-1}Pg \mid g \in G\}$ を考える. G の Ω への作用は推移的であるから, 本節の始めに述べたように, 剰余類 $\Omega_1 = St_G(x)\backslash G$ への G の右移動による作用と同等である (ただし $x \in \Omega$). ここでは Ω_1 への作用を使うことにする. まず $x = P$ として, $St_G(x) = N_G(P)$ に注意する. $N_G(P)$ は P を含む G の部分群であるが, 実はこの場合, $N_G(P) = P$ となる.

```
gap> Normalizer(a8, syl2) = syl2;
true
```

そこで, 作用域 Ω_1 =omega1 を右剰余類 $P\backslash G$ =RightCosets(a8,sly2) として定義し, H =elab3 の Ω_1 への右作用による軌道分解を考える.

```
gap> omega1 := RightCosets(a8,syl2);;
gap> obd := OrbitLengthsDomain(elab3, omega1, OnRight);
[ 8, 8, 2, 8, 2, 8, 2, 2, 8, 8, 4, 4, 8, 1, 8, 4, 4, 8, 8, 8, 8, 4, 1, 4, 2,
  2, 1, 8, 2, 2, 2, 8, 2, 8, 4, 1, 8, 4, 4, 8, 4, 1, 8, 4, 1, 2, 2, 2, 2, 2,
  4, 1, 4, 4, 1, 4, 1, 1, 8, 8, 4, 4, 2, 1, 8, 4, 2, 1, 2, 1, 4, 2, 1, 4, 2,
  1, 1, 4, 1, 1, 2, 1, 1, 1 ]
gap> Size(obd);
84
```

かくして Ω_1 は 84 個の H-軌道 $\mathcal{O}_1, \ldots, \mathcal{O}_{84}$ に分解した. これは G の P と H による両側剰余類 $P\backslash G/H$ が 84 個あると言い替えることもできる. いくつかの軌道を具体的に求めてみよう.

```
gap> dcosets := OrbitsDomain(elab3, omega1, OnRight);;
gap> SetName(syl2, "P2");
gap> dcosets[3];
[ RightCoset(P2,(6,8,7)), RightCoset(P2,(1,3)(2,4)(5,7,8)) ]
gap> dcosets[14];
[ RightCoset(P2,(4,5,7,8,6)) ]
```

3 番目の軌道 $\mathcal{O}_3$ は 2 個の元からなり, 14 番目の軌道 $\mathcal{O}_{14}$ は 1 個の元からなる. Ω_1 の元を Ω の元として読み換えて, G の 2-シロー群を作ると,

```
gap> IsSubgroup(syl2^(6,8,7), elab3);
false
gap> IsSubgroup(syl2^(4,5,7,8,6), elab3);
true
```

$\mathcal{O}_{14}$ から, H を含む 2-シロー群 $g^{-1}Pg$ $(g = (4,5,7,8,6))$ が得られた. 軌道の集合 dcosets から, 長さ 1 の軌道のみを取り出すには, 2.6 節に述べた Filtered(,) を使う.

```
gap> sylsets := Filtered(dcosets, c -> Size(c) = 1);
[ [ RightCoset(P2,(4,5,7,8,6)) ], [ RightCoset(P2,(4,7,6)) ],
  [ RightCoset(P2,(3,5)(4,7)) ], [ RightCoset(P2,(2,3)(4,5,6)(7,8)) ],
  [ RightCoset(P2,(2,3)(4,6)) ], [ RightCoset(P2,(2,3,5)(4,6,7)) ],
  [ RightCoset(P2,(2,4,5,6,3)) ], [ RightCoset(P2,(2,4,6,3)(7,8)) ],
  [ RightCoset(P2,(2,4,6,8,7,3,5)) ], [ RightCoset(P2,(2,5,4,3)(7,8)) ],
  [ RightCoset(P2,(2,5,3)) ], [ RightCoset(P2,(2,5)(6,7)) ],
  [ RightCoset(P2,(2,6,5,4,3)) ], [ RightCoset(P2,(2,6,5,3)(7,8)) ],
  [ RightCoset(P2,(2,6,8,7,5)) ], [ RightCoset(P2,(2,7,6,8,5,4,3)) ],
  [ RightCoset(P2,(2,7,5,3)(6,8)) ], [ RightCoset(P2,(2,7,8,6,5)) ],
  [ RightCoset(P2,(2,8,6,7,5,4,3)) ], [ RightCoset(P2,(2,8,5,3)(6,7)) ],
  [ RightCoset(P2,(2,8,5)) ] ]
gap> Size(sylsets);
21
```

21 個の長さ 1 の軌道が得られた. これらから, 直前に述べたプロセスにより P の共役部分群 $g^{-1}Pg$ を作ると, それが H =elab3 を含む G の 2-シロー群をすべて (21 個) 与える.

最後に G =a8 の 2 シロー部分群の総数を求めておこう. 2-シロー群はすべて G-共役であるから $\Omega = \{g^{-1}Pg \mid g \in G\}$ の元の個数が 2-シロー群の総数である. (5.6.2) より $|\Omega| = |G|/|St_G(P)|$. $St_G(P) = N_G(P) = P$ だったから $|\Omega| = |G|/|P|$ である.

```
gap> Size(a8)/Size(syl2);
315
```

これより A_8 の 2 シロー群の総数は, $|G|/|P| = 20160/64 = 315$ となる. (5.7.2) (ii) は, 総数が $\mod 2$ で 1 であることを要請しているが, この場合 $(p = 2)$ 奇数になるので当然成立する.

5.8 共役類

群 G の元 g, g' は $h^{-1}gh = g'$ となる $h \in G$ が存在するとき, 共役であるといい $g \sim g'$ と表す. $g \sim g'$ は同値関係であり, その同値類を**共役類** (conjugacy class) という. $x^{-1}gx$ を g^x とも表す. そこで $g \in G$ を含む共役類は $g^G = \{x^{-1}gx \mid x \in G\}$ である. 共役類とは G の G への共役の作用に関する軌道に他ならない. したがって, G は共通部分のない共役類の和集合 (disjoint union) に分解される.

$$G = \coprod_{g \in G/\sim} g^G$$

ただし, $G/\sim$ は G の共役類 (の代表元) の集合を表す. 特に,

$$|G| = \sum_{g \in G/\sim} |g^G| \tag{5.8.1}$$

が成り立つ. 各 $g \in G$ に対し, $Z_G(g) = \{x \in G \mid x^{-1}gx = g\}$ を g の**中心化群** (centralizer) という. 共役の作用の場合, $g \in G$ に対し, $St_G(g) = Z_G(g)$ である. そこで, (5.6.2) により, 各 $g \in G$ に対して

$$|G| = |g^G||Z_G(g)|$$

が成立する. (5.8.1) と合せて

$$1 = \sum_{g \in G/\sim} \frac{1}{|Z_G(g)|} \tag{5.8.2}$$

が得られる. (5.8.1) または (5.8.2) を有限群 G の**類等式**という.

GAP では, 群 G の共役類への分解は, `ConjgacyClasses(G)` により軌道分解を経由せずに直接求められる. 例えば交代群 $G = A_8$ の共役類のリスト `ccl` は次のように与えられる.

```
gap> ccl := ConjugacyClasses( a8 );
[ ()^G, (1,2)(3,4)^G, (1,2)(3,4)(5,6)(7,8)^G, (1,2,3)^G,
  (1,2,3)(4,5)(6,7)^G, (1,2,3)(4,5,6)^G, (1,2,3,4)(5,6)^G,
  (1,2,3,4)(5,6,7,8)^G, (1,2,3,4,5)^G, (1,2,3,4,5)(6,7,8)^G,
  (1,2,3,4,5)(6,8,7)^G, (1,2,3,4,5,6)(7,8)^G,
  (1,2,3,4,5,6,7)^G, (1,2,3,4,5,6,8)^G ]
gap> Length (ccl);
14
```

G は 14 個の共役類に分解される. GAP は各共役類を代表元をひとつ決めて, 例えば $g^G =$ `(1,2)(3,4)^G` のように表示するが, 共役類の代表元だけを取り出したかったら, 例によって

```
gap> rep := List (ccl, c -> Representative (c));
[ (), (1,2)(3,4), (1,2)(3,4)(5,6)(7,8), (1,2,3),
  (1,2,3)(4,5)(6,7), (1,2,3)(4,5,6), (1,2,3,4)(5,6),
  (1,2,3,4)(5,6,7,8), (1,2,3,4,5), (1,2,3,4,5)(6,7,8),
  (1,2,3,4,5)(6,8,7), (1,2,3,4,5,6)(7,8),
  (1,2,3,4,5,6,7), (1,2,3,4,5,6,8) ]
```

とすればよい. 各共役類が何個の元からなるかを知るには,

```
gap> List(ccl, Size);
[ 1, 210, 105, 112, 1680, 1120, 2520, 1260, 1344, 1344,
  1344, 3360, 2880, 2880 ]
```

`List(ccl, Size)` はリスト `ccl` の各要素（これもリスト）に, そのリストとしてのサイズを対応させてリストを作る命令である.

```
gap> Sum(List(ccl,Size)) = Size(a8);
true
```

によって (5.8.1) が確かめられる. 上のリストで例えば 3 番目の共役類は, 105 個の元からなる集合である. その元を全て表示したければ,

```
gap> Elements(ccl[3]);
```

とすればよい. ここには結果を書かないが, 興味のある人は自分でやってみて下さい.

GAP では元 $g \in G$ の中心化群 $Z_G(g)$ は `Centralizer(G, g)` により与えられる. 例えば 2 番目の共役類 g^G ($g = (1,2)(3,4)$) については $|g^G| = 210$ であり,

```
gap> Centralizer(a8, (1,2)(3,4));
Group([ (6,7,8), (5,6)(7,8), (3,4)(7,8), (1,2)(7,8), (1,3)(2,4) ])
gap> Size(Centralizer(a8, (1,2)(3,4)));
96
gap> 96 * 210 = Size(a8);
true
```

これより $|g^G||Z_G(g)| = |G|$ も確かめられる. 各代表元 $g \in G/\sim$ に対する $1/|Z_G(g)|$ のリストは

```
gap> inv:= List(rep, c -> 1/Size(Centralizer(a8,c)));
[ 1/20160, 1/96, 1/192, 1/180, 1/12, 1/18, 1/8, 1/16,
  1/15, 1/15, 1/15, 1/6, 1/7, 1/7 ]
```

によって得られる. そこで (5.8.2) 式が確かめられる.

```
gap> Sum(inv);
1
```

具体的にこの式を書き下してみると,

$$1 = \frac{1}{20160} + \frac{1}{96} + \frac{1}{192} + \frac{1}{180} + \frac{1}{12} + \frac{1}{18} + \frac{1}{8} + \frac{1}{16} + \frac{1}{15} + \frac{1}{15} + \frac{1}{15} + \frac{1}{6} + \frac{1}{7} + \frac{1}{7}$$

最初の項の分母の大きさから考えて, この式はかなりショッキングな関係式である. 試みに $1/20160$ に分数 $1/N$ (N は 20160 の約数) をいくつか少ない個数を加えて総和を 1 にすることを考えてみればよい. そう簡単な話ではないことが理解されるだろう. 類等式は簡単な等式ながら有限群の持つ隠された対称性を如実に表現する関係式なのである.

5.2 節に述べたように, 群 G の各元 g に対して位数が定まる. G のすべての元の位数を決定するにはどうしたらいいだろうか. ひとつの共役類に含まれる元の位数は一定であることに気がつくと, この問題は簡単に解決する. 各共役類の代表元の位数が分ればすべての G の元の位数が分るのである. 各共役類に含まれる元の位数のリストは次のようにして得られる.

```
gap> List(ccl, c -> Order(Representative(c)));
[ 1, 2, 2, 3, 6, 3, 4, 4, 5, 15, 15, 6, 7, 7 ]
```

10 番目, 12 番目の共役類の代表元は,

```
gap> Representative(ccl[10]); Representative(ccl[12]);
(1,2,3,4,5)(6,7,8)
(1,2,3,4,5,6)(7,8)
```

であるから, 確かに位数はそれぞれ, 15, 6 になっている.

5.9 対称群の共役類と分割数

本節では, 対称群 S_n の共役類を分類しよう. S_n の元 g は共通な文字を持たない巡回置換の積として (順番を無視すれば) 一意的に表される. GAP では, そもそもどんな元もこのような形で表示されているのであった.

```
gap> g := (1,2,3)*(2,5,6)*(3,1,4);
(1,5,6,2)(3,4)
```

これは, g で生成される巡回群 $G = \langle g \rangle$ の 集合 $\Omega = \{1, 2, 3, \dots\}$ への作用に関する軌道分解に他ならない.

```
gap> G := Group(g);
Group([ (1,5,6,2)(3,4) ])
gap> OrbitsDomain(G);
[ [ 1, 5, 6, 2 ], [ 3, 4 ] ]
```

g は 6 文字 $\{1, 2, \dots, 6\}$ の置換なので, 何も断らなければ GAP は自動的に $\Omega = \{1, 2, \dots, 6\}$ と解釈する. g を S_8 の元と考えて, 作用域を $\Omega = \{1, 2, \dots, 8\}$ と指定すれば,

```
gap> OrbitsDomain(G, [1..8]);
[ [ 1, 5, 6, 2 ], [ 3, 4 ], [ 7 ], [ 8 ] ]
gap> OrbitLengthsDomain(G, [1..8]);
[ 4, 2, 1, 1 ]
```

ここに出て来た $(4, 2, 1, 1)$ を S_8 の元 g の**サイクル・タイプ**という. サイクル・タイプは, 軌道の元の個数を大きい順に並べたものである. $\Omega = \{1, 2, \dots, 8\}$ であるから当然 $4 + 2 + 1 + 1 = 8$ となる. このようにサイクル・タイプを考えるとき, 1 も勘定に入れることが重要である. その意味で, g を S_8 の元として考える場合, 単に $g = (1, 5, 6, 2)(3, 4)$ ではなく, 不動点も考慮して, $g = (1, 5, 6, 2)(3, 4)(7)(8)$ と表すこともある. 一般に, S_n の元 g のサイクル・タイプは

$$\lambda = (\lambda_1, \lambda_2, \dots, \lambda_r)$$

と表される. ここに $\{\lambda_i\}$ は $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0$ かつ, $\sum_{i=1}^{r} \lambda_i = n$ を満たす自然数である. このような自然数の組 λ を n の**分割** (partition) という. 分割 λ を表すのに以下のような表記法を使うことも多い.

$$\lambda = (\lambda_1, \dots, \lambda_r) = (1^{m_1}, 2^{m_2}, \dots, n^{m_n})$$

後半の表示は, λ のパーツが, 1 が m_1 個, 2 が m_2 個, ... からできていることを示している. すなわち, $\sum_{i=1}^{n} im_i = n$. 例えば,

$$(4,3,3,1) = (1^1, 2^0, 3^2, 4^1)$$

さて S_n の元 g のサイクル・タイプは n の分割 λ を定めると言ってもよい. S_n の共役類はサイクル・タイプによって完全に特徴付けられる. すなわち

(5.9.1) $g, g' \in S_n$ が共役になるための必要十分条件は g と g' が同じサイクル・タイプを持つことである. したがって, S_n の共役類の集合と n の分割の集合は 1 対 1 に対応する.

実際, 長さ k の巡回置換 $g = (a_1, a_2, \ldots, a_k)$ と $\sigma \in S_n$ に対し, $\sigma^{-1}g\sigma$ は再び長さ k の巡回置換 $\sigma^{-1}g\sigma = (a_1\sigma, a_2\sigma, \ldots, a_k\sigma)$ になる. これから容易に, 一般の $g \in S_n$ に対して, $\sigma^{-1}g\sigma$ は g と同じサイクル・タイプを持つことが導かれる. 逆に, $g, g' \in S_n$ が同じサイクル・タイプを持つとき, 上の議論を参考にして $\sigma^{-1}g\sigma = g'$ となる $\sigma \in S_n$ を見付けることができる. 以下に例を示す. 一般の場合は例えば,

$$g = (2,1,3,4)(7,6,8)(5), \quad g' = (8,6,1,2)(4,3,5)(7)$$

を S_8 の元 (サイクル・タイプは共に $(4,3,1)$) とすると,

$$\sigma = \begin{pmatrix} 2 & 1 & 3 & 4 & 7 & 6 & 8 & 5 \\ 8 & 6 & 1 & 2 & 4 & 3 & 5 & 7 \end{pmatrix} = (1,6,3)(2,8,5,7,4)$$

とすればよい. 念のために GAP で検算してみると,

```
gap> g := (2,1,3,4)(7,6,8)(5);
(1,3,4,2)(6,8,7)
gap> s := (1,6,3)*(2,8,5,7,4);
(1,6,3)(2,8,5,7,4)
gap> g^s;
(1,2,8,6)(3,5,4)
```

表示は異なるが, これは `g^s` が $g' = (8,6,1,2)(4,3,5)(7)$ に一致することを示している. GAP には, 群 G のふたつの元 x, y が G 上共役かどうかを判定する命令 `IsConjugate(G,x,y)` が用意されている.

```
gap> g1 := (8,6,1,2)(4,3,5)(7);
(1,2,8,6)(3,5,4)
gap> IsConjugate(s8, g, g1);
true
```

(5.9.1) により, S_n の共役類の分類は n の分割の分類という純粋に組合せ論的な問題に帰着する. $\mathcal{P}_n$ を n の分割の集合とする. $\lambda \in \mathcal{P}_n$ に対し, 対応する $G = S_n$ の共役類を C_λ と表し, $g_\lambda \in C_\lambda$ をその代表元とする. C_λ の元の個数は $|C_\lambda| = |G|/|Z_G(g_\lambda)|$ で与えられるので, 中心化群 $Z_G(g_\lambda)$ の位数を知ることが重要である. 次が成り立つ.

(5.9.2) $\lambda \in \mathcal{P}_n$ を $\lambda = (1^{m_1}, 2^{m_2}, \ldots, n^{m_n})$ と表す. このとき

$$|Z_G(g_\lambda)| = \prod_{i=1}^{n} m_i! \cdot i^{m_i}.$$

(5.9.2) を示そう．$g_\lambda = \prod_{i=1}^{n}\prod_{j=1}^{m_i} g_{ij}$ と表せる．ただし g_{ij} は長さ i の巡回置換で、互いに共通の文字を持たないものとする．(5.9.1) の証明の議論から, $\sigma \in \S_n$ が $Z_G(g_\lambda)$ に含まれるための条件は, $i = 1, \ldots, n$ に対して集合 $X_i = \{g_{i,1}, \ldots, g_{i,m_i}\}$ が σ の共役の作用で不変になることである．そこで $\sigma = \sigma_1 \ldots \sigma_n$ と表される．ただし $\sigma_i \in S_{im_i}$．σ_i は X_i の置換を自由に取れる, それは $m_i!$ の可能性がある．σ_i の X_i への置換を一つ固定する．例えば, 恒等置換 $\sigma_i^{-1} g_{ij} \sigma_i = g_{ij}$ とすれば, $\sigma_i = \prod_{j=1}^{m_i} g_{ij}^{k_j}$ の形に書ける．ここに $0 \le k_j \le i-1$ は任意に取れる．したがって, X_i の置換を固定したとき, σ_i には i^{m_i} の取り方ができる．これより (5.9.2) が得られる．

具体例で調べてみよう．$\lambda = (3,3,2) = (1^0, 2^1, 3^2)$ とすれば, $g_\lambda \in S_8$ として $g_\lambda = (1,2,3)(4,5,6)(7,8)$ が取れる．このとき (5.9.2) により

$$|Z_G(g_\lambda)| = (2! \cdot 3^2) \times (1! \cdot 2^1) = 36$$

一方, GAP で計算してみると

```
gap> g := (1,2,3)(4,5,6)(7,8);
(1,2,3)(4,5,6)(7,8)
gap> Size(Centralizer(s8, g));
36
```

となり, めでたしめでたしである．

$\mathcal{P}_n$ の元の個数を n の**分割数** といい, $p(n)$ と表す．$\mathcal{P}_n$ も $p(n)$ も GAP で簡単に求められる．分割 $\lambda = (\lambda_1, \ldots, \lambda_r)$ に対し, そのパーツの個数 r を λ の長さといい, $r = l(\lambda)$ と表す．また $n = \sum \lambda_i$ を $n = |\lambda|$ と書く．さて, $|\lambda| = n$, $l(\lambda) = k$ となる分割の集合を $\mathcal{P}_{n,k}$ と表す．$\mathcal{P}_{n,k}$ は, `Partitions(n,k)` で与えられる．k を省略し単に `Partitions(n)` と書けば n の分割の集合 $\mathcal{P}_n$ が得られる．

```
gap> Partitions(3);
[ [ 1, 1, 1 ], [ 2, 1 ], [ 3 ] ]
gap> Partitions(4);
[ [ 1, 1, 1, 1 ], [ 2, 1, 1 ], [ 2, 2 ], [ 3, 1 ], [ 4 ] ]
gap> Partitions(8,3);
[ [ 3, 3, 2 ], [ 4, 2, 2 ], [ 4, 3, 1 ], [ 5, 2, 1 ], [ 6, 1, 1 ] ]
```

$p(n)$ は `NrPartitions(n)` で求められる．集合 $\mathcal{P}_{n,k}$ の元の個数は `NrPartitions(n,k)` である．

```
gap> NrPartitions(3);
3
gap> NrPartitions(8,3);
5
```

関数 $p(n)$ は非常に興味深い関数である．試みに, $n = 1 \sim 100$ に対して $p(n)$ を計算してみると,

```
gap> part := List([1..100], x -> NrPartitions(x));
[ 1, 2, 3, 5, 7, 11, 15, 22, 30, 42, 56, 77, 101, 135, 176, 231, 297, 385,
  490, 627, 792, 1002, 1255, 1575, 1958, 2436, 3010, 3718, 4565, 5604, 6842,
  8349, 10143, 12310, 14883, 17977, 21637, 26015, 31185, 37338, 44583, 53174,
  63261, 75175, 89134, 105558, 124754, 147273, 173525, 204226, 239943,
  281589, 329931, 386155, 451276, 526823, 614154, 715220, 831820, 966467,
  1121505, 1300156, 1505499, 1741630, 2012558, 2323520, 2679689, 3087735,
  3554345, 4087968, 4697205, 5392783, 6185689, 7089500, 8118264, 9289091,
  10619863, 12132164, 13848650, 15796476, 18004327, 20506255, 23338469,
  26543660, 30167357, 34262962, 38887673, 44108109, 49995925, 56634173,
  64112359, 72533807, 82010177, 92669720, 104651419, 118114304, 133230930,
  150198136, 169229875, 190569292 ]
```

$p(100) = 190569292$ は既に 1 億を突破しており, $p(n)$ が急速に増大する関数であることが分かるだろう. `Partitions(100)` を GAP に計算させるようなことは厳につつしまなければならない. さてこの表を眺めてみると最初の数項は素数がならんでいることに気がつくだろう. $p(2) = 2, p(3) = 3, p(4) = 5, p(6) = 7, p(7) = 11$. ここで, $p(4) = 5$ から始めて 5 個ずつ先を取っていくと

```
gap> part[4]; part[9]; part[14];
5
30
135
```

すべて 5 で割り切れることが分かる. 本格的に確かめてみると,

```
gap> part5 := List([4,9..99], x -> NrPartitions(x));
[ 5, 30, 135, 490, 1575, 4565, 12310, 31185, 75175, 173525,
  386155, 831820, 1741630, 3554345, 7089500, 13848650,
  26543660, 49995925, 92669720, 169229875 ]
```

`[4,9..99]` が 2.4 節で説明した, 初項 4, 公差 5, 最終項 99 の等差数列を変域とせよという指定である. 確かに, $p(5k + 4)$ がすべて 5 で割り切れることが分かる. 次に, $p(5) = 7$ に注目して 7 個ずつ増やしていくと

```
gap> part[5]; part[12]; part[19];
7
77
490
```

やはり, すべて 7 で割り切れる. 前と同じように残りも調べてみると

```
gap> part7 := List([5,12..96], x -> NrPartitions(x));
[ 7, 77, 490, 2436, 10143, 37338, 124754, 386155,
  1121505, 3087735, 8118264, 20506255, 49995925, 118114304 ]
```

これは, ちょっと見ただけでは 7 で割れるかどうか分からない. そこで 7 で割った余りを取り出してみると

```
gap> List(part7, x -> x mod 7);
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
```

見事に 7 の倍数になっていることが分かる. 実は同様のことが $p(6) = 11$ についても成立するのである.

```
gap> part11 := List([6,17..94], x -> NrPartitions(x));
[ 11, 297, 3718, 31185, 204226, 1121505, 5392783, 23338469, 92669720 ]
gap> List(part11, x -> x mod 11);
[ 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
```

関数 $p(n)$ について以下の等式が一般に成立することが Ramanujan により示されている. これらを Ramanujan の等式という.

$$\begin{aligned} p(5k+4) &\equiv 0 \pmod 5, \\ p(7k+5) &\equiv 0 \pmod 7, \\ p(11k+6) &\equiv 0 \pmod{11} \end{aligned}$$

また, $p(n)$ の近似的な値については, Hardy-Ramanujan により

$$p(n) \sim \frac{1}{4\sqrt{3}n} \exp(\pi\sqrt{2n/3})$$

という驚くべき公式が知られている. Ramanujan の等式や, Hardy-Ramanujan の公式など, $p(n)$ の深い性質を研究するための出発点になるのが, 以下に述べる母関数の考え方である. 一般に数列 $\{a_n\}_{n=0}^{\infty}$ が与えられたとき,

$$f(t) = \sum_{n=0}^{\infty} a_n t^n$$

とおいて, $f(t)$ を数列 $\{a_n\}$ の**母関数** (generating function) という. $f(t)$ は t を不定元とする形式的巾級数として考える. $a_n \in \mathbb{C}$ の場合には, 収束域を定めて実際の関数として扱うことも多い. 数列 $\{a_n\}$ の性質は母関数 $f(x)$ にすべて反映されているとみるのが母関数の立場である. 個々の a_n は一見ばらばらで, 扱いに手を焼いても, それらを全体として一段上の立場から見れば, 驚く程素直で統制が取れているということは, よくあることであろう. 一般項 a_n が決められなくても関数 $f(x)$ が記述できれば, それを利用して, a_n の性質を導くことができる. 母関数は, うまくはまった時, 信じられないような威力を発揮するのである. 母関数は generating function の訳語であるから, 直訳すれば生成関数である. しかし, 生成関数ではいかにも味気ないではないか. 私は, 数列 $\{a_n\}$ の母なる関数としての思いが込められている母関数という言葉が大好きである.

さて, $p(0) = 1$ として, $p(n)$ の母関数

$$F(t) = \sum_{n=0}^{\infty} p(n) t^n$$

を考えよう. $p(n)$ は S_n の共役類の個数だったから, $F(t)$ は S_n の共役類の母関数といってもよい. とりあえずここでは, $F(t)$ を整数係数の形式的巾級数として扱う. 次が成り立つ.

$$F(t) = \prod_{n=1}^{\infty}(1-t^n)^{-1} \tag{5.9.3}$$

(5.9.3) を示そう.

$$\begin{aligned}\prod_{i=1}^{\infty}\frac{1}{1-t^i} =&(1+t+t^2+\cdots+t^{m_1}+\cdots)\times\\ &(1+t^2+(t^2)^2+\cdots+t^{2m_2}+\cdots)\times\\ &(1+t^3+(t^3)^2+\cdots+t^{3m_3}+\cdots)\times\\ &\cdots\cdots\cdots\\ &(1+t^i+(t^i)^2+\cdots+t^{im_i}+\cdots)\times\\ &\cdots\cdots\cdots\end{aligned}$$

右辺を展開したときの一般項は, $t^{m_1+2m_2+\cdots im_i+\cdots}$ で与えられる. そこで右辺を整理すれば, t^n の係数は $\sharp\{(m_1,\ldots,m_n) \mid \sum_{i=1}^{n} im_i = n\} = p(n)$ となる. したがって (5.9.3) が成立する.

さて (5.9.3) の $F(t)$ は複素関数とみるとき, $|t|<1$ で絶対収束し, 正則関数を与える. $q=\exp(2pi\tau)$ とするとき, Dedekind の η 関数 $\eta(\tau)=\zeta\prod_{n=1}^{\infty}(1-q^n)$ (ζ は 1 の 24 乗根) を通じて $F(t)$ の性質が調べられ, そこから $p(n)$ に関する性質が導かれる. 純粋に組合せ論的な対象であった分割数が母関数を仲立ちにして, 楕円関数論の大海に躍り出ていくのである.

せっかく $p(n)$ の母関数 $F(t)$ が定まったので, これを利用して $p(n)$ を計算してみよう. そのためには, GAP に多項式の計算をやらせなければならない. GAP で不定元 t に関する有理関数 $f(t)\in\mathbb{Q}(t)$ を定義するには

```
gap> t := Indeterminate(Rationals, "t");
t
```

とすればよい. これで多項式や有理式の計算が自由にできる. 例えば

```
gap> (t +1)^5;
t^5+5*t^4+10*t^3+10*t^2+5*t+1
gap> (1-t)^-1;
(1)/(-t+1)
```

与えられた多項式 $P(t)$ の係数だけを取り出してリストを作るには,

```
gap> P := t^5 + 3*t^2 + 2*t + 1;
t^5+3*t^2+2*t+1
gap> CoefficientsOfUnivariatePolynomial(P);
[ 1, 2, 3, 0, 0, 1 ]
```

とする. UnivariatePolynomial とは 1 変数の多項式という意味である. 係数は 0 次から最高次まで昇巾の順に並んでいる. $F(t)$ を使って $n = 1 \sim 100$ までの $p(n)$ を計算してみよう. $P_i(t) = (1-t^i)^{-1}$ とおく. $F(t) = \prod_{i\geq 1} P_i(t)$ である. $F(t)$ を t^{100} まで計算したいので, 各 $P_i(t) = 1 + t^i + \cdots t^{im_i} + \cdots$ を $im_i < 100$ となる最大の m_i まで計算する必要がある. そこで $m_i = [100/i]$ とおき ($[x]$ は x を越えない最大の整数),

$$\widetilde{P}_i(t) = 1 + t^i + \cdots + t^{im_i} = \frac{1-t^{i(m_i+1)}}{1-t^i}$$

に対して, $\prod_{i=1}^{100} \widetilde{P}_i(t)$ の展開における 100 次までの項を調べればよい. まず $m_1, \ldots, m_{100}$ のリスト mm を計算する.

```
gap> mm := List([1..100], i -> (100-(100 mod i))/i );
[ 100, 50, 33, 25, 20, 16, 14, 12, 11, 10, 9, 8, 7, 7, 6, 6, 5, 5,
  5, 5, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ]
```

次に $\widetilde{P}_1(t), \ldots, \widetilde{P}_{100}(t)$ のリスト factors を計算する.

```
gap> factors := List([1..100], i -> (1-t^(i*(mm[i]+1)))/(1-t^i));;
```

これは打ち出させると大変なことになる. いくつか書いてみると

```
gap> factors[1];
t^100+t^99+t^98+t^97+t^96+t^95+t^94+t^93+t^92+t^91+t^90+t^89+t^88+t^87+t^86+
t^85+t^84+t^83+t^82+t^81+t^80+t^79+t^78+t^77+t^76+t^75+t^74+t^73+t^72+t^71+
t^70+t^69+t^68+t^67+t^66+t^65+t^64+t^63+t^62+t^61+t^60+t^59+t^58+t^57+t^56+
t^55+t^54+t^53+t^52+t^51+t^50+t^49+t^48+t^47+t^46+t^45+t^44+t^43+t^42+t^41+
t^40+t^39+t^38+t^37+t^36+t^35+t^34+t^33+t^32+t^31+t^30+t^29+t^28+t^27+t^26+
t^25+t^24+t^23+t^22+t^21+t^20+t^19+t^18+t^17+t^16+t^15+t^14+t^13+t^12+t^11+
t^10+t^9+t^8+t^7+t^6+t^5+t^4+t^3+t^2+t+1
gap> factors[2];
t^100+t^98+t^96+t^94+t^92+t^90+t^88+t^86+t^84+t^82+t^80+t^78+t^76+t^74+t^72+
t^70+t^68+t^66+t^64+t^62+t^60+t^58+t^56+t^54+t^52+t^50+t^48+t^46+t^44+t^42+
t^40+t^38+t^36+t^34+t^32+t^30+t^28+t^26+t^24+t^22+t^20+t^18+t^16+t^14+t^12+
t^10+t^8+t^6+t^4+t^2+1
gap> factors[3];
t^99+t^96+t^93+t^90+t^87+t^84+t^81+t^78+t^75+t^72+t^69+t^66+t^63+t^60+t^57+
t^54+t^51+t^48+t^45+t^42+t^39+t^36+t^33+t^30+t^27+t^24+t^21+t^18+t^15+t^12+
t^9+t^6+t^3+1
```

次に partfcn $= \prod_{i=1}^{100} \widetilde{P}_i(t)$ を計算して

```
gap> partfcn := Product(factors);;
```

さらに得られた多項式 partfcn の係数を並べたリストを作る.

```
gap> coeff := CoefficientsOfUnivariatePolynomial(partfcn);;
gap> Size(coeff);
8300
```

係数のリストのサイズは, 8300 つまり, partfcn は 8299 次というとんでもない巨大な多項式になる. さて coeff[1] は 0 次の係数, coeff[2] は 1 次の係数, etc. であるから, リスト coeff から, 2.6 節のようにして, 第 2 項から第 101 項までを抜き出したリストを作る.

```
gap> cpart := coeff{[2..101]};;
```

これが求める $p(1) \sim p(100)$ のリストを与えるはずである. 実際確かめてみると

```
gap> part = cpart;
true
```

となり, 見事に一致することが分かる. それにしても何という非効率的な計算だろう. 我々は 100 次までの係数を求めるのに, 8299 次の多項式を計算したのである. GAP のダイナミズムに支えられて始めて可能になったことで手計算では膨大な時間を消費することになる. 母関数による方法の理論面での強力さと, 実際の計算における非力さのコントラストには, 驚くばかりである.

5.10 交代群の共役類とその母関数

対称群 S_n の共役類については良く分かったので, 今度は交代群 A_n の共役類について調べよう. $\widetilde{G} = S_n$, $G = A_n$ とおく. G は $\widetilde{G}$ の正規部分群なので, G は $\widetilde{G}$ の共役類の和集合になる. C_λ を分割 $\lambda \in \mathcal{P}_n$ に対応する $\widetilde{G}$ の共役類とする. $g_\lambda \in C_\lambda$ を代表元とすると, $g_\lambda \in G$ ならば $C_\lambda \subset G$, $g_\lambda \notin G$ ならば $C_\lambda \cap G = \emptyset$ である. $\lambda = (\lambda_1, \ldots, \lambda_r)$ とすると, g_λ は長さ $\lambda_1, \lambda_2, \ldots, \lambda_r$ の巡回置換の積になる. ところで, 長さ k の巡回置換は $k-1$ 個の互換の積で書ける. したがって, g_λ は $\sum_{i=1}^{r}(\lambda_i - 1) = n - l(\lambda)$ 個の互換の積で書ける. すなわち, $l(\lambda) \equiv n \pmod 2$ のとき, $g_\lambda \in G$, そうでなければ $g_\lambda \notin G$ である. ここで

$$\mathcal{P}_n^0 = \{\lambda \in \mathcal{P}_n \mid l(\lambda) \equiv n \pmod 2\},$$
$$\mathcal{P}_n^1 = \{\lambda \in \mathcal{P}_n \mid l(\lambda) \not\equiv n \pmod 2\}$$

とおく. 当然 $\mathcal{P}_n = \mathcal{P}_n^0 \cup \mathcal{P}_n^1$ である. 今までの議論から

$$G = \coprod_{\lambda \in \mathcal{P}_n^0} C_\lambda \tag{5.10.1}$$

となる. C_λ は G の共役の作用で不変なので, C_λ は G の共役類の和集合に分解する. この分解のしかたが分かれば, $G = A_n$ の共役類は分類できる. それを知るために, 5.8 節で調べた A_8 の共役類を S_8 の共役類と比較してみよう.

```
gap> s8class := ConjugacyClasses(s8);;
gap> reps8 := List(s8class, c -> Representative (c));;
gap> repa8 := Filtered(reps8, c -> c in a8);
[ (), (1,2)(3,4), (1,2)(3,4)(5,6)(7,8), (1,2,3), (1,2,3)(4,5)(6,7),
  (1,2,3)(4,5,6), (1,2,3,4)(5,6), (1,2,3,4)(5,6,7,8), (1,2,3,4,5),
  (1,2,3,4,5)(6,7,8), (1,2,3,4,5,6)(7,8), (1,2,3,4,5,6,7) ]
```

S_8 の共役類のリストを求め, その代表元からなるリストを作り, そこから A_8 に含まれる元だけを取り出してリスト repa8 を作った. 上の表に対応する n の分割を並べれば次のようになる.

$$(1^8),\quad (2^2,1^4),\quad (2^4),\quad (3,1^5),\quad (3,2^2,1),\quad (3^2,1^2),$$
$$(4,2,1^2),\quad (4^2),\quad (5,1^3),\quad (5,3),\quad (6,2),\quad (7,1)$$

これは, 8 の分割で, (5.10.1) の条件を満たすものの全体である. 実際

```
gap> part8 := Partitions(8);;
gap> Filtered(part8, c -> Size(c) mod 2 = 0);
[ [ 1, 1, 1, 1, 1, 1, 1, 1 ], [ 2, 2, 1, 1, 1, 1 ], [ 2, 2, 2, 2 ],
  [ 3, 1, 1, 1, 1, 1 ], [ 3, 2, 2, 1 ], [ 3, 3, 1, 1 ], [ 4, 2, 1, 1 ],
  [ 4, 4 ], [ 5, 1, 1, 1 ], [ 5, 3 ], [ 6, 2 ], [ 7, 1 ] ]
```

ここで, repa8 と A_8 の共役類の代表元のリスト rep を比べてみると, 以下の事実が分かる. $\lambda \in \mathcal{P}_8^0$ に対し, $\lambda = (5,3)$ または $\lambda = (7,1)$ のとき, C_λ は $C_\lambda = C'_\lambda \cup C''_\lambda$ と A_8 の 2 つの共役類に分解する. $\lambda = (5,3)$ のとき, C'_λ, C''_λ の代表元は, それぞれ $(1,2,3,4,5)(6,7,8), (1,2,3,4,5)(6,8,7)$ で与えられる. $\lambda = (7,1)$ のとき, C'_λ, C''_λ の代表元は $(1,2,3,4,5,6,7), (1,2,3,4,5,6,8)$ で与えられる. $\lambda \notin \{(5,3),(7,1)\}$ については, C_λ が A_8 の共役類を与える. 実際 GAP で確かめてみると,

```
gap> rests8 := Filtered(s8class, c -> Representative(c) in a8);;
gap> List(rests8, c -> OrbitLengthsDomain(a8, c));
[ [ 1 ], [ 210 ], [ 105 ], [ 112 ], [ 1680 ], [ 1120 ], [ 2520 ], [ 1260 ],
  [ 1344 ], [ 1344, 1344 ], [ 3360 ], [ 2880, 2880 ] ]
```

S_8 の共役類のリストから, A_8 に含まれるもの rests8 を取り出し, 各共役類の上に, A_8 を共役で作用させたものを軌道分解してリストを作った. 確かに, $\lambda = (5,3)$ と $\lambda = (7,1)$ に対応する C_λ は同じサイズの 2 つの軌道 [1344, 1344], [2880,2880] に分解していることが分かる.

以上の事実を一般の $G = A_n$ について確かめよう. まず, $\lambda \in \mathcal{P}_n^0$ に対して, C_λ は 高々2 個の G-共役類に分解することに注意する. 実際, $\widetilde{G} = G \cup \sigma G$ (σ は $\widetilde{G}$ のある元) と書けるので,

$$C_\lambda = g_\lambda^{\widetilde{G}} = g_\lambda^G \cup (g_\lambda^\sigma)^G = C'_\lambda \cup C''_\lambda$$

となる. ここに C'_λ, C''_λ は G の共役類で, g_λ と g_λ^σ が G 上共役ならば $C'_\lambda = C''_\lambda = C_\lambda$, そうでなければ $C'_\lambda \neq C''_\lambda$ である. この事はまた中心化群 $Z_G(g_\lambda)$ の言葉で次のように言い替えることもできる.

(5.10.2) $Z_G(g_\lambda)$ は $Z_{\widetilde{G}}(g_\lambda)$ の高々指数 2 の部分群であり, $[Z_{\widetilde{G}}(g_\lambda) : Z_G(g_\lambda)] = 2$ ならば, C_λ は G の共役類に一致し, $Z_{\widetilde{G}}(g_\lambda) = Z_G(g_\lambda)$ ならば, C_λ は G 上 2 個の同じサイズの共役類に分解する.

実際, $|C_\lambda| = |\widetilde{G}|/|Z_{\widetilde{G}}(g_\lambda)|$, $|C'_\lambda| = |G|/|Z_G(g_\lambda)| = |\widetilde{G}|/(2|Z_G(g_\lambda)|)$ よりこれはすぐ出る.

そこで, (5.10.2) において $Z_{\widetilde{G}}(g_\lambda) = Z_G(g_\lambda)$ となるための $\lambda \in \mathcal{P}_n^0$ の条件を求めよう. これは, $Z_{\widetilde{G}}(g_\lambda)$ が奇置換を含まないということと同値である. (5.9.2) の $Z_{\widetilde{G}}(g_\lambda)$ の計算を思い出そう. $\lambda = (\lambda_1, \ldots, \lambda_r)$ とするとき, λ_i に対応する巡回置換は $Z_{\widetilde{G}}(g_\lambda)$ の元を与える. もし λ_i が偶数なら対応する巡回置換は奇置換になる. したがって $\lambda_1, \ldots, \lambda_r$ はすべて奇数でなければならない. さらに, 各パーツの重複度は 1 である. 実際, もしある λ_i が重複しているとする, 例えば $\lambda_1 = \lambda_2 = k$ とすると, $g_\lambda = (a_1, \ldots, a_k)(b_1, \ldots, b_k)\cdots$ と表される. このとき, $\sigma = (a_1, b_1)(a_2, b_2)\cdots(a_k, b_k)$ は g_λ と可換になる. しかし, σ は奇数個の互換の積なので奇置換になり, 最初の仮定に反する. 以上により, $Z_{\widetilde{G}}(g_\lambda) = Z_G(g_\lambda)$ ならば, $\lambda_1, \ldots, \lambda_r$ は互いに相異なる奇数であることが分かった. 逆に $\lambda_1, \ldots, \lambda_r$ がすべて異なれば, (5.9.2) により $Z_{\widetilde{G}}(g_\lambda \simeq \mathbb{Z}/\lambda_1\mathbb{Z} \times \cdots \times \mathbb{Z}/\lambda_r\mathbb{Z}$ であり, 各 $\mathbb{Z}/\lambda_i\mathbb{Z}$ の生成元は, 長さ λ_i (奇数) の巡回置換である. したがって, $Z_{\widetilde{G}}(g_\lambda)$ の元はすべて偶置換となる. 以上をまとめて次が得られる.

(5.10.3) $n \geq 2$ とする. ($n = 1$ の場合は, $S_1 = A_1 = \{1\}$).

(i) S_n の共役類 C_λ が A_n に含まれる条件は $\lambda \in \mathcal{P}_n^0$ である.

(ii) $\lambda = (\lambda_1, \ldots, \lambda_r) \in \mathcal{P}_n^0$ とする. C_λ が A_n の 2 個の共役類に分解するのは, $\lambda_1, \ldots, \lambda_r$ がすべて相異なる奇数の場合に限る. (この場合, 同じサイズの 2 つの共役類に分解する). その他の場合 C_λ は常に A_n の共役類に一致する.

A_n の共役類の個数を $a(n)$ とおく. $n = 2, \ldots, 100$ に対して $a(n)$ を求めてみよう. $\mathcal{P}_n^0$ の個数を $b(n)$ とおく. また (5.10.3) の (ii) を満たす λ の集合を $\mathcal{P}_n^{00}$ と表し, その元の個数を $c(n)$ とおく. (5.10.3) より

$$a(n) = b(n) + c(n) \tag{5.10.4}$$

が成り立つ. まず $c(n)$ から調べていこう. $n = 2 \sim 10$ に対して $c(n)$ および $\mathcal{P}_n^{00}$ は次のようになる.

n	$c(n)$	$\mathcal{P}_n^{00}$
2	0	
3	1	(3)
4	1	(3,1)
5	1	(5)
6	1	(5,1)
7	1	(7)
8	2	(7,1), (5,3)
9	2	(9), (5,3,1)
10	2	(9,1), (7,3)

一般の $c(n)$ を記述するには, 例によって母関数を使うのが有効である. 便宜上, $c(0) = 1, c(1) = 1$ とおいて,

$$H_1(t) = \sum_{n=0}^{\infty} c(n)t^n$$

とおく. この場合は簡単に $H_1(t)$ が求まる.

$$H_1(t) = \prod_{n=1}^{\infty}(1 + t^{2n-1}) \tag{5.10.5}$$

$H_1(t)$ を使って $n=2\sim100$ までの $c(n)$ を計算してみよう. $H_1(t)$ の右辺の積を $n=1\sim50$ まで計算すれば十分である. まず, 各因子のリスト factor1 を定義する.

```
gap> factor1 := List ([1..50], i -> 1 + t^(2*i-1));;
gap> factor1[3];
t^5+1
```

次に積 singfcn $=\prod_{n=1}^{50}(1+t^{2n-1})$ を計算して, その係数を取り出す.

```
gap> singfcn := Product(factor1);;
gap> coeff1 := CoefficientsOfUnivariatePolynomial(singfcn);;
gap> Size(coeff1);
2501
```

3 番目が $c(2)$, 101 番目が $c(100)$ なので, $3\sim101$ の係数を取り出せば $n=2,\dots,100$ に対する $c(n)$ のリストが得られる.

```
gap> singpart := coeff1{[3..101]};
[ 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7, 8, 8, 9, 11, 12,
  12, 14, 16, 17, 18, 20, 23, 25, 26, 29, 33, 35, 37, 41, 46, 49, 52, 57, 63,
  68, 72, 78, 87, 93, 98, 107, 117, 125, 133, 144, 157, 168, 178, 192, 209,
  223, 236, 255, 276, 294, 312, 335, 361, 385, 408, 437, 471, 501, 530, 568,
  609, 647, 686, 732, 784, 833, 881, 939, 1004, 1065, 1126, 1199, 1279, 1355,
  1433, 1523, 1621, 1717, 1814, 1925, 2048, 2166, 2286, 2425, 2574 ]
```

次に $b(n)=|\mathcal{P}_n^0|$ を調べよう. $n=2\sim9$ に対しては $b(n)$ および $\mathcal{P}_n^0$ は次のようになる.

n	$b(n)$	$\mathcal{P}_n^0$
2	1	(1^2)
3	2	$(3),(1^3)$
4	3	$(3,1),(2^2),(1^4)$
5	4	$(5),(3,1^2),(2^2),(1^4)$
6	6	$(5,1),(4,2),(3^2),(3,1^3),(2^2,1^2),(1^6)$
7	8	$(7),(5,1^2),(4,2,1),(3^2,1),(3,2^2),(3,1^4),(2^2,1^3),(1^7)$
8	12	$(7,1),(6,2),(5,3),(5,1^3),(4^2),(4,2,1)$
		$(3^2,1^2),(3,2^2,1),(3,1^5),(2^4),(2^2,1^4),(1^8)$
9	16	$(9),(7,1^2),(6,2,1),(5,3,1),(5,2^2),(5,1^4),(4^2,1),(4,3,2)$
		$(4,2,1^3),(3^3),(3^2,1^3),(3,2^2,1^2),(3,1^6),(2^4,1),(2^2,1^5),(1^9)$

(5.10.1) により, 一般の $b(n)$ は次の式で与えられる.

$$b(n)=\sum_{\substack{1\le k\le n\\ k\equiv n \pmod 2}}|\mathcal{P}_{n,k}|$$

ここで $|\mathcal{P}_{n,k}|$ は前に述べたように NrPartitions(n,k) により計算できる. そこで, この命令を利用して各 n に対して $b(n)$ を 与える関数 $b(n)=$altpart(n) を作ろう.

```
gap> altpart := function(n)
>      local k, nr, c;
>      nr := 0;
>      c := (n+1) mod 2 + 1;
>      for k in [c, c+2..n] do
>          nr := nr + NrPartitions(n,k);
>      od;
>      return nr;
>    end;
function( n ) ... end
gap> altpart(3); altpart(4); altpart(8);
2
3
12
```

これより $b(n)$ を $n = 2 \sim 100$ まで並べたリスト normalpart が得られる.

```
gap> normalpart := List([2..100], i -> altpart(i));
[ 1, 2, 3, 4, 6, 8, 12, 16, 22, 29, 40, 52, 69, 90, 118, 151, 195, 248, 317,
  400, 505, 632, 793, 985, 1224, 1512, 1867, 2291, 2811, 3431, 4186, 5084,
  6168, 7456, 9005, 10836, 13026, 15613, 18692, 22316, 26613, 31659, 37619,
  44601, 52815, 62416, 73680, 86809, 102162, 120025, 140853, 165028, 193144,
  225710, 263490, 307161, 357699, 416006, 483338, 560864, 650196, 752877,
  870953, 1006426, 1161916, 1340012, 1544048, 1777365, 2044188, 2348821,
  2696627, 3093095, 3545015, 4059416, 4644850, 5310255, 6066425, 6924691,
  7898630, 9002580, 10253568, 11669704, 13272332, 15084211, 17132044,
  19444436, 22054694, 24998640, 28317803, 32056941, 36267714, 41005947,
  46335767, 52326672, 59058176, 66616548, 75100211, 84616150, 95285933 ]
```

$a(n) = b(n) + c(n)$ だったから, これより A_n の共役類の個数 $a(n)$ を $n = 2 \sim 100$ まで並べたリスト altclass が得られる.

```
gap> altclass := normalpart + singpart;
[ 1, 3, 4, 5, 7, 9, 14, 18, 24, 31, 43, 55, 72, 94, 123, 156, 200, 254, 324,
  408, 513, 641, 804, 997, 1236, 1526, 1883, 2308, 2829, 3451, 4209, 5109,
  6194, 7485, 9038, 10871, 13063, 15654, 18738, 22365, 26665, 31716, 37682,
  44669, 52887, 62494, 73767, 86902, 102260, 120132, 140970, 165153, 193277,
  225854, 263647, 307329, 357877, 416198, 483547, 561087, 650432, 753132,
  871229, 1006720, 1162228, 1340347, 1544409, 1777750, 2044596, 2349258,
  2697098, 3093596, 3545545, 4059984, 4645459, 5310902, 6067111, 6925423,
  7899414, 9003413, 10254449, 11670643, 13273336, 15085276, 17133170,
  19445635, 22055973, 24999995, 28319236, 32058464, 36269335, 41007664,
  46337581, 52328597, 59060224, 66618714, 75102497, 84618575, 95288507 ]
```

ここで, S_n の場合にならって, 数列 $\{a(n)\}$ の母関数を求めてみよう. $a(0)=1, a(1)=1$ とおき,

$$H(t)=\sum_{i=0}^{\infty} a(n)t^n$$

とおく. (5.10.4) により, $H(t)=H_0(t)+H_1(t)$ と表される. ただし ($b(0)=1, b(1)=1$ として),

$$H_0(t)=\sum_{n=0}^{\infty} b(n)t^n,$$

$H_1(t)$ は前に定義した $c(n)$ の母関数である. (5.10.5) により, $H_1(t)$ は既に分かっているので, $H_0(t)$ を決めればよい. $H_0(t)$ については次が成り立つ.

$$H_0(t)=\frac{1}{2}\prod_{n=1}^{\infty}\frac{1}{1-t^{2n-1}}\left(\prod_{n=1}^{\infty}\frac{1}{1-t^{2n}}+\prod_{n=1}^{\infty}\frac{1}{1+t^{2n}}\right) \tag{5.10.6}$$

(5.10.6) を示そう. $\mathcal{P}_n^0$ に λ が入るための条件をもう少し詳しく見てみる. $\lambda=(\lambda_1,\ldots,\lambda_r)=(1^{m_1},2^{m_2},\ldots,n^{m_n})$ とするとき, λ_k が奇数ならば 対応する巡回置換は偶置換であり, $g_\lambda\in A_n$ となる条件に影響しない. そこで $g_\lambda\in A_n$ となる条件は, 長さ偶数の巡回置換のすべての積が A_n に入ることである. 言い替えれば, $\lambda\in\mathcal{P}_n^0$ となる条件は,

$$\sum_{i:\text{even}} m_i\equiv 0 \pmod 2 \tag{5.10.7}$$

で与えられる. さて (5.9.3) の議論を思い出すと, $\prod_{i:\text{odd}}(1-t^i)^{-1}$ の展開からは, $\sum t^{\sum im_i}$ (i は奇数を動く) が得られ, $\prod_{i:\text{even}}(1-t^i)^{-1}$ の展開からは $\sum t^{\sum im_i}$ で i が偶数を動くものが得られる. 一方, $\prod_{i:\text{even}}(1+t^i)^{-1}$ の展開からは, $\sum(-1)^{\sum m_i}t^{\sum im_i}$ (i は偶数を動く) が得られる. したがって, $\prod_{i:\text{even}}(1-t^i)^{-1}+\prod_{i:\text{even}}(1+t^i)^{-1}$ の展開からは, $\sum 2t^{\sum im_i}$ (最初の和は (5.10.7) を満たす $(m_1,\ldots,m_n)$ を動く. 2 番目の和は偶数 i を動く) が得られる. これより (5.10.6) が導かれる.

(5.10.5) とあわせて交代群 A_n の共役類の母関数 $H(t)$ が決定される. ($c(0), c(1)$ は実際とずれているので, $n\geq 2$ に対する t^n の係数が $a(n)$ を与える).

$$H(t)=\frac{1}{2}\prod_{n=1}^{\infty}\frac{1}{1-t^{2n-1}}\left(\prod_{n=1}^{\infty}\frac{1}{1-t^{2n}}+\prod_{n=1}^{\infty}\frac{1}{1+t^{2n}}\right)+\prod_{n=1}^{\infty}(1+t^{2n-1}) \tag{5.10.8}$$

最後に $H_0(t)$ を使って, $n=2\sim 100$ に対する $b(n)$ を計算してみよう. $H_0(t)=((F(t)+F_1(t))/2$ と表される. ただし, $F(t)$ は 5.9 節で述べた $p(n)$ の母関数, また $F_1(t)$ は

$$F_1(t)=\prod_{n=1}^{\infty}\frac{1}{1-t^{2n-1}}\prod_{n=1}^{\infty}\frac{1}{1+t^{2n}}$$

で与えられる. $F(t)$ は既に計算したので, $F_1(t)$ を計算すればよい. $F(t)$ を計算したときのやり方を踏襲する. $i=1,\ldots,100$ までを奇数と偶数に分けて, $F_1(t)$ の奇数因子のリストと偶数因子のリストを定義する.

```
gap> oddfac := List([1,3..99], i -> (1 - t^(i*(mm[i]+1)))/(1 - t^i));;
gap> evenfac := List([2,4..100], i -> (1 - (-t^i)^(mm[i]+1))/(1 + t^i));
```

奇数因子と偶数因子のすべての積 product を作り, その係数からなるリストを作る.

```
gap> product := Product(oddfac)*Product(evenfac);;
gap> altcoef := CoefficientsOfUnivariatePolynomial(product);;
gap> Size(altcoef);
8300
```

係数のリスト altcoef から, 2 次から 100 次までを取り出したリスト c2part を作る.

```
gap> c2part := altcoef{[3..101]};
[ 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7, 8, 8, 9,
  11, 12, 12, 14, 16, 17, 18, 20, 23, 25, 26, 29, 33, 35, 37, 41,
  46, 49, 52, 57, 63, 68, 72, 78, 87, 93, 98, 107, 117, 125, 133,
  144, 157, 168, 178, 192, 209, 223, 236, 255, 276, 294, 312, 335,
  361, 385, 408, 437, 471, 501, 530, 568, 609, 647, 686, 732, 784,
  833, 881, 939, 1004, 1065, 1126, 1199, 1279, 1355, 1433, 1523,
  1621, 1717, 1814, 1925, 2048, 2166, 2286, 2425, 2574 ]
```

cpart は既に計算してあるが (1 次から 100 次まで), 次数をそろえるために, もう一度定義する.

```
gap> c1part := coeff{[3..101]};;
```

$H_0(t) = (F(t) + F_1(t))/2$ なので, c1part と c2part の和の半分が $H_0(t)$ の係数 normalpart を与えるはずである.

```
gap> (c1part + c2part)/2 = normalpart;
true
```

となり, 一致した. めでたし, めでたし.

と, これで終るはずだったのだが, まだ続くのである. c2part の表を見て頂きたい. なぜこの表に負の整数が現れないのだろうか. 多項式の因子にはマイナスも入っているのでこれは不思議なことである. そう思ってこの表をじっくり眺めてみると, なにやらこれは見覚えのある表である. 賢明な読者は既に気づかれたことと思うが, この表は前に計算した singpart の表とそっくり同じである. 念のため, 確かめてみると

```
gap> singpart = c2part;
true
```

この奇妙な一致は次の恒等式からの帰結である.

$$\prod_{n=1}^{\infty} \frac{1}{1-t^{2n-1}} \prod_{n=1}^{\infty} \frac{1}{1+t^{2n}} = \prod_{n=1}^{\infty} (1+t^{2n-1}) \tag{5.10.9}$$

(5.10.9) を示そう. 明らかに (5.10.9) は

$$\prod_{n=1}^{\infty}\frac{1}{1-t^{2n-1}}=\prod_{n=1}^{\infty}(1+t^n)$$

と同値である. 一方, この式の右辺は

$$\prod_{n=1}^{\infty}(1+t^n)=\prod_{n=1}^{\infty}\frac{1-t^{2n}}{1-t^n}=\prod_{n:\mathrm{odd}}\frac{1}{1-t^n}$$

と変形できる. 2 番目の等式は, 分母の t の偶数次の因子がすべて分子とキャンセルされて, t の奇数次の因子のみが分母に残ることから得られる. したがって (5.10.9) が成り立つ.

さて (5.10.9) 式は, $H_1(t)=F_1(t)$ を意味する. これより (5.10.8) は

$$H(t)=\frac{1}{2}F(t)+\frac{3}{2}H_1(t)=\frac{1}{2}\prod_{n=1}^{\infty}\frac{1}{1-t^n}+\frac{3}{2}\prod_{n=1}^{\infty}(1+t^{2n-1}) \tag{5.10.10}$$

とすっきりした形で表される.

ところで, 恒等式 (5.10.9) は, その係数を比較することによって, 次のような興味深い関係式を導く. 分割 $\lambda=(1^{m_1},2^{m_2},\ldots,n^{m_n})$ に対し,

$$\Delta(\lambda)=\sum_{i:\mathrm{even}}m_i$$

とおく. このとき,

$$\sharp\{\lambda\in\mathcal{P}_n\mid\Delta(\lambda):\mathrm{even}\}-\sharp\{\lambda\in\mathcal{P}_n\mid\Delta(\lambda):\mathrm{odd}\}=|\mathcal{P}_n^{00}| \tag{5.10.11}$$

が成り立つ. $\{\lambda\in\mathcal{P}_n\mid\Delta(\lambda):\mathrm{even}\}=\mathcal{P}_n^0$, $\{\lambda\in\mathcal{P}_n\mid\Delta(\lambda):\mathrm{odd}\}=\mathcal{P}_n^1$ であることに注意しよう. $|\mathcal{P}_n^0|=b(n)$, $|\mathcal{P}_n^{00}|=c(n)$ であった. ここで $b'(n)=|\mathcal{P}_n^1|$ とおく. 明らかに $b(n)+b'(n)=p(n)$ が成り立つ. $p(n),b(n),c(n)$ は既に計算してあるので, これより $b'(n)$ を求め, (5.10.11) 式すなわち, $b(n)-b'(n)=c(n)$ を確かめてみよう. $b(n)$ は $n=2\sim100$ だったので, $p(n)$ のリスト `part` を変形して `ppart` を作り, `ppart - normalpart` として $b'(n)$ のリスト `conormalpart` を作る.

```
gap> ppart := List([2..100], c -> NrPartitions(c));;
gap> conormalpart := ppart - normalpart;
[ 1, 1, 2, 3, 5, 7, 10, 14, 20, 27, 37, 49, 66, 86, 113, 146, 190, 242, 310,
  392, 497, 623, 782, 973, 1212, 1498, 1851, 2274, 2793, 3411, 4163, 5059,
  6142, 7427, 8972, 10801, 12989, 15572, 18646, 22267, 26561, 31602, 37556,
  44533, 52743, 62338, 73593, 86716, 102064, 119918, 140736, 164903, 193011,
  225566, 263333, 306993, 357521, 415814, 483129, 560641, 649960, 752622,
  870677, 1006132, 1161604, 1339677, 1543687, 1776980, 2043780, 2348384,
  2696156, 3092594, 3544485, 4058848, 4644241, 5309608, 6065739, 6923959,
  7897846, 9001747, 10252687, 11668765, 13271328, 15083146, 17130918,
  19443237, 22053415, 24997285, 28316370, 32055418, 36266093, 41004230,
  46333953, 52324747, 59056128, 66614382, 75097925, 84613725, 95283359 ]
```

$b(n)-b'(n)=c(n)$ を調べるには, `normalpart - conormalpart` が `singpart` と一致することを見ればよい.

```
gap> normalpart - conormalpart = singpart;
true
```

確かに成立する. 参考のため以下に $n=1\sim 20$ までの $p(n),a(n),b(n),b'(n),c(n)$ の表を載せておく.

n	$p(n)$	$a(n)$	$b(n)$	$b'(n)$	c(n)
1	1	1	1	0	1
2	2	1	1	1	0
3	3	3	2	1	1
4	5	4	3	2	1
5	7	5	4	3	1
6	11	7	6	5	1
7	15	9	8	7	1
8	22	14	12	10	2
9	30	18	16	14	2
10	42	24	22	20	2
11	56	31	29	27	2
12	77	43	40	37	3
13	101	55	52	49	3
14	135	72	69	66	3
15	176	94	90	86	4
16	231	123	118	113	5
17	297	156	151	146	5
18	385	200	195	190	5
19	490	254	248	242	6
20	627	324	317	310	7

定義を思い出しておくと $p(n)$ が, S_n の共役類の個数, $a(n)$ が A_n の共役類の個数, また

$$
\begin{aligned}
b(n) &= |\mathcal{P}_n^0| = \ A_n \text{ に含まれる } S_n \text{ の共役類の個数}\\
b'(n) &= |\mathcal{P}_n^1| = A_n \text{ に含まれない } S_n \text{ の共役類の個数}\\
c(n) &= |\mathcal{P}_n^{00}| = A_n \text{ への制限が}2\text{つの共役類に分裂するような } S_n \text{ の共役類の個数}
\end{aligned}
$$

である. $p(n),a(n),b(n),b'(n),c(n)$ の間には次の関係式が成立する.

$$
\begin{aligned}
p(n) &= b(n)+b'(n), \quad (n\geq 1)\\
c(n) &= b(n)-b'(n), \quad (n\geq 1)\\
a(n) &= b(n)+c(n), \quad (n\geq 2)
\end{aligned}
\tag{5.10.12}
$$

したがって $a(n)=3b(n)-p(n)$ $(n\geq 2)$. $p(n),b(n)$ は関数として定義されているので, これより $a(n)$ を与える関数 `altclass(n)` $=a(n)$ を作ることができる. ($n=1$ の場合は補正をする).

```
gap> altclass := function(n)
>       local c;
>       if n = 1 then c := 1;
>          else c := 3*altpart(n) - NrPartitions(n);
```

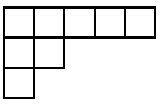
```
>       fi;
>       return c;
>    end;
function( n ) ... end
gap> altclass(1); altclass(10); altclass(100); altclass(200);
1
24
95288507
1986499984086
```

ところでさっき作った表をちらちら眺めてみると, $p(n), a(n), b(n)$ などと, $c(n)$ の増大度の違いが印象的である. 多分 $\lim_{n\to\infty} c(n)/b(n) = 0$ であろう. (5.10.12) より, $p(n) = 2b(n) - c(n)$ であるから, これより

$$\lim_{n\to\infty} \frac{A_n \text{ の共役類の個数}}{S_n \text{ の共役類の個数}} = \frac{1}{2} \tag{5.10.13}$$

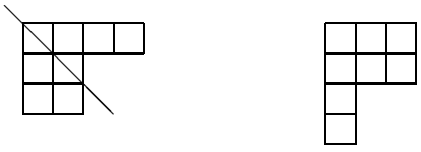
という, 何やらもっともらしい予想が得られる.

分割 $\lambda = (\lambda_1, \ldots, \lambda_r)$ を視覚的に表現するものとして**ヤング図形** (Young diagram) がよく知られている. ヤング図形とは, 平面上に n 個の 4 角の箱を, 横に λ_1 個, その下に λ_2 個, ... と左側をそろえて並べたものである. 例えば, $\lambda = (5, 2, 1)$ に対応するヤング図形は次のようになる. λ に対応するヤン



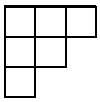
$\lambda = (5, 2, 1)$

グ図形を主対角線で折り返して出来るヤング図形に対応する分割を λ^* と表わし, λ の**双対分割** (dual partition) という. 例えば, $\lambda = (4, 2, 2)$ に対し $\lambda^* = (3, 3, 1, 1)$ である.



$\lambda = (4, 2, 2)$ $\qquad$ $\lambda^* = (3, 3, 1, 1)$

$\mathcal{P}_n^\natural = \{\lambda \in \mathcal{P}_n \mid \lambda = \lambda^*\}$ とおく. $\mathcal{P}_n^\natural$ は双対分割が自分自身と一致するような n の分割の集合である. 例えば, $\lambda = (3, 2, 1)$ は $\mathcal{P}_n^\natural$ に含まれる.



$\lambda = (3,2,1)$

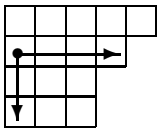
以下に $n = 1 \sim 10$ までの $\mathcal{P}_n^\natural$ の表をあげておこう.

n	$\lvert\mathcal{P}_n^\natural\rvert$	$\mathcal{P}_n^\natural$
1	1	(1)
2	0	—
3	1	(2,1)
4	1	(2,2)
5	1	(3,1,1)
6	1	(3,2,1)
7	1	(4,1,1,1)
8	2	(4,2,1,1), (3,3,2)
9	2	(5,1,1,1,1), (3,3,3)
10	2	(5,2,1,1,1), (4,3,2,1)

$c(n) = |\mathcal{P}_n^{00}|$ の表と比べると, $n = 10$ までは, $c(n) = |\mathcal{P}_n^\natural|$ となっている. 実際この 2 つの数は一致するのである. (前の表では $c(n)$ は $n \geq 2$ でしか書いてないが, $\mathcal{P}_n^{00}$ は $n = 1$ についても定義できて $c(1) = 1$ になる.)

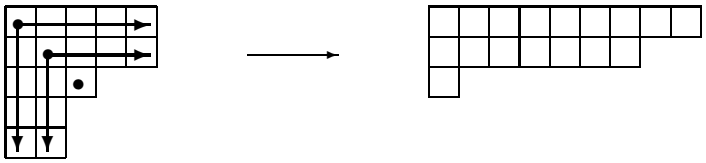
(5.10.14) $$|\mathcal{P}_n^{00}| = |\mathcal{P}_n^\natural|$$

実際簡単に $\mathcal{P}_n^\natural$ から $\mathcal{P}_n^{00}$ への全単射を作ることができる. その説明のためにヤング図形の**フック** (hook) を定義しよう. まずヤング図形に座標を入れ, 左上隅の箱を (1,1) 成分, 第 i 行 j 列の箱を (i,j) 成分と名前をつける. (i,j) を足にするフックとは, ヤング図形の中で, (i,j) 成分の右側にある箱と下側にある箱 ((i,j) も含める) を取りだしてできる鈎型の図形のことをいう. フックに含まれる箱の数をフックの長さという. 下の図では (2,1) 成分を足とする長さ 6 のフックを描いている.



(5.10.14) の説明に戻ろう. $\lambda \in \mathcal{P}_n^\natural$ とする. λ のヤング図形は主対角線に対して対称だから (i,i) 成分に関するフックの長さは常に奇数である. また, λ のヤング図形から, $(1,1)$ 成分のフック, $(2,2)$ 成分のフックと順次取り出していくと, ヤング図形は互いに重なることのないフックの和集合として表される. 第 (i,i) 成分のフックの長さを h_i とすると $\mu = (h_1, h_2, \dots)$ は n の分割になる. $\mu \in \mathcal{P}_n^{00}$ であり, $\lambda \mapsto \mu$ が全単射 $\mathcal{P}_n^\natural \to \mathcal{P}_n^{00}$ を与える. 次に例をあげておく.

$|\mathcal{P}_n^{00}| = |\mathcal{P}_n^\natural|$ が成り立つ背景には次のような表現論的な意味合いがある. S_n の既約表現は $\mathcal{P}_n$ によってラベル付けされる. $\lambda \in \mathcal{P}_n$ に対応する S_n の既約表現を ρ_λ と表す. クリフォード理論により, A_n の既約表現はすべて S_n の既約表現を分解することにより得られる. ρ_λ の A_n への制限を $\rho_\lambda|_{A_n}$ と表す.



$\lambda = (5,5,3,2,2,)$ $\mu = (9,7,1)$

このとき $\rho_\lambda|_{A_n} = \rho_{\lambda'}|_{A_n}$ となる条件は $\lambda' = \lambda^*$ となることであり, $\lambda \neq \lambda^*$ ならば $\rho_\lambda|_{A_n}$ は既約になる. また $\lambda = \lambda^*$ ならば $\rho_\lambda|_{A_n}$ はふたつの異なる既約表現の直和に分解する. そこで, $\mathcal{P}_n^\natural$ からふたつづつの既約表現が得られ, $\mathcal{P}_n - \mathcal{P}_n^\natural$ から丁度半分の既約表現が得られる. $\mathcal{P}_n - \mathcal{P}_n^\natural$ に対応する A_n の既約表現は S_n の制限なので S_n の表現論を利用して調べることができる. 一方, $\mathcal{P}_n^\natural$ に対応する A_n の既約表現には A_n に特有の性質が現れて, 調べるのが難しい.

ところで, 一般に有限群の共役類の個数と既約表現の個数は一致することが知られている. これらのふたつの集合の間に標準的な全単射が与えられるわけではないが、多くの場合, 共役類と既約表現の間には単なる全単射以上の, 親密で微妙な関係が成立しているのである. S_n の共役類も $\mathcal{P}_n$ によりラベル付けられる. S_n の共役類 C_λ が A_n と共通部分を持つのは, $\lambda \in \mathcal{P}_n^0$ の場合である. このうち $\lambda \in \mathcal{P}_n^0 - \mathcal{P}_n^{00}$ ならば, C_λ の A_n への制限 (つまり共通部分) はそのまま A_n の共役類を与える. 一方, $\lambda \in \mathcal{P}_n^{00}$ ならば, この制限は A_n のふたつの共役類に分解する. ここでも A_n に特有の現象は $\lambda \in \mathcal{P}_n^{00}$ で現れる. このような状況を考えれば, $|\mathcal{P}_n^{00}|$ と $|\mathcal{P}_n^\natural|$ とは, どうしても一致してもらわなくては困るのである. そうでなければ, 性格の不一致といった問題に発展しかねないではないか. というわけで, $|\mathcal{P}_n^{00}| = |\mathcal{P}_n^\natural|$ が確かめられて, 我々はやれやれ良かったと胸をなでおろすのであった.

$|CP_n^{00}| = |\mathcal{P}_n^\natural|$ は分かったが, その他の部分では共役類と既約表現の関係はどうなっているのだろうか. A_n の残りの既約表現の個数は $(|\mathcal{P}_n| - |\mathcal{P}_n^\natural|)/2 = (p(n) - c(n))/2$ で与えられる. 一方, A_n の共役類で, $\mathcal{P}_n^{00}$ を除いたものの個数は $|\mathcal{P}_n^0| - |\mathcal{P}_n^{00}| = b(n) - c(n)$ で与えられる. したがって, 両者は一致するはずである.

$$\frac{1}{2}(p(n) - c(n)) = b(n) - c(n) \tag{5.10.15}$$

実際, (5.10.12) より, $p(n) + c(n) = 2b(n)$ が成り立ち, (5.10.15) が得られる. また, この式より, $c(n) = 2b(n) - p(n)$. $p(n), b(n)$ は関数として得られているので, これより $c(n)$ も関数として計算できることを注意しておく.

5.11 p-シロー群の構成

5.7 節で保留にしてあった p-シロー群の存在を示しておこう. $|G| = p^m k$, $(p,k) = 1$ とする. 今, $\mathcal{M}$ を G の p^m 個の元からなる部分集合全体の集合とする. 当然,

$$|\mathcal{M}| = \binom{kp^m}{p^m}$$

である. G は右移動により集合 $\mathcal{M}$ に作用する. 例によって $\mathcal{M}$ を G-軌道に分解する.

$$\mathcal{M} = \coprod_{A \in \mathcal{M}/G} \mathcal{O}_A$$

このとき, $\mathcal{M}$ には必ず $|\mathcal{O}_A|$ が p と互いに素となるような軌道 $\mathcal{O}_A$ が存在する. 実際これは, 組合せ論の等式

$$\binom{qp^m}{p^m} \equiv q \pmod{p} \tag{5.11.1}$$

から出る. (ここで q は任意の自然数). (5.11.1) より $q = k$ とおいて $|\mathcal{M}|$ が p と素であることが分かる. そこでもしすべての軌道について $|\mathcal{O}_A|$ が p で割り切れるならば, $|\mathcal{M}|$ も p で割り切れてしまう. これは矛盾である. (5.11.1) 式自体は, 有限体 $\mathbb{F}_q$ 上の等式 $(1+t)^{p^m q} = (1+t^{p^m})^q$ の両辺を 2 項展開することにより得られる. さて $A \in \mathcal{M}$ を $|\mathcal{O}_A|$ が p と素になるような G の部分集合とする. このとき, A の固定化群 $St_G(A)$ は G のひとつの p-シロー群を与えるのである. 実際, $|G| = |\mathcal{O}_A||St_G(A)|$ より, $|St_G(A)|$ は p^m で割り切れる. 一方, $St_G(A)$ は A に置換として作用する. この作用は G の右移動から得られる作用なので, $a \in A$ を固定すると, 写像 $St_G(A) \to A, g \to ag$ は単射になる, すなわち $|St_G(A)| \leq |A| = p^m$. これより $|St_G(A)| = p^m$ が得られ, $St_G(A)$ が G の p-シロー群になることが分かった.

(5.11.1) はここでは証明しないが, GAP でいくつかの実例を計算して納得してもらおう. GAP では 2 項係数 $\binom{a}{b}$ は `Binomial(a,b)` により計算できる.

```
gap> Binomial(3*8,8); Binomial(5*16,16);
735471
26958221130508525
gap> (Binomial(7*5^3,5^3)-7) mod 5; (Binomial(6*7^4,7^4)-6) mod 7;
0
0
```

さて, 上記の方法にしたがって実際に p-シロー群を構成してみよう. $G = A_4$, $p = 2$ とする. ずいぶん小さい群でみみっちくやるものだと不興を買うかも知れないが, これにはよんどころない事情があるのである. 事情は後回しにして, とにかくやってみよう. $|A_4| = 12$ より, $p^m = 4$ である. そこで 4 個の元からなる $G =$`a4` の部分集合全体の集合 `dom` $= \mathcal{M}$ を考える. それには, 以前出てきた `Combinations` という命令を使う. これが群の元に対しても有効なのが強みである.

```
gap> dom := Combinations(Set(a4), 4);;
```

G を集合として扱うので, `Set(a4)` としなければならないことに注意する. 次に $\mathcal{M}$ の上への G の作用を定義する. これがまあ, 何とも悩ましいのである. 試みに `dom[1]` に右作用で $g = (1,3)(2,4)$ を作用させてみると,

```
gap> dom[1];
[ (), (2,3,4), (2,4,3), (1,2)(3,4) ]
gap> OnRight(dom[1], (1,3)*(2,4));
[ (1,3)(2,4), (1,3,2), (1,3,4), (1,4)(2,3) ]
```

ちょっと見づらいが, `OnRight` によって得られた最後のリストは集合になっていない. 実際,

```
gap> Set(last);
[ (1,3,2), (1,3,4), (1,3)(2,4), (1,4)(2,3) ]
```

となり, これは前のものと順番が変わっている. つまり, OnRight の作用では, $A \in \mathcal{M}$ の各元にそのまま, G が作用するだけであり, Ag を集合として扱うためには得られたリストを並べ直さなければならないのである. 集合として扱いたいのであるから, それでは OnSets を使ったらどうだろうか.

```
gap> OnSets(dom[1], (1,3)*(2,4));
[ (), (1,2)(3,4), (1,2,4), (1,4,2) ]
```

これでは, 先程のものと全く違ったものが出来てしまう. この場合, A の各元に g が共役で作用してしまったのである. これでは役に立たない. という訳で, GAP の用意してくれた作用では, 今の場合, 間に合わないのである. ここは, どうしても頑張って自前の作用を作らなければならない. GAP には, 新しい作用を定義する方法も用意されている. 一般に G の集合 X への作用は, 写像 $f : X \times G \to X$ によって与えられる. そこで, GAP では $x \in X$, $g \in G$ に対し, 関数 actfcn(x,g) を定義すればよい.

```
gap> OnSetsRight := function(set, g)
> return Set(set*g);
> end;
function( set, g ) ... end
```

これで, 群の右移動の後, 並べ直して集合として扱うという新しい作用 OnSetsRight が定義できた.

```
gap> OnSetsRight(dom[1], (1,3)*(2,4));
[ (1,3,2), (1,3,4), (1,3)(2,4), (1,4)(2,3) ]
```

確かに求める作用になっている. そこで前のように, dom を G-軌道に分解する.

```
gap> OrbitLengthsDomain(a4, dom, OnSetsRight);
[ 12, 12, 12, 12, 6, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
  12, 12, 12, 12, 12, 12, 12, 12, 6, 12, 12, 12, 12, 12, 12, 12,
  12, 12, 12, 6, 12, 6, 12, 6, 12, 3, 12, 6, 12 ]
gap> Size(last);
45
```

$\mathcal{M}$ =dom は 45 個の G-軌道に分解した. この中に長さ奇数の軌道が少なくとも一つ存在するはずである. 確かに長さ 3 の軌道が見つかる. これを取り出すことにしよう.

```
gap> orb := OrbitsDomain(a4, dom, OnSetsRight);;
gap> list := Filtered(orb, c -> Size(c) mod 2 = 1);
[ [ [ (), (1,2)(3,4), (1,3)(2,4), (1,4)(2,3) ],
    [ (2,4,3), (1,2,3), (1,3,4), (1,4,2) ],
    [ (2,3,4), (1,2,4), (1,3,2), (1,4,3) ] ] ]
```

これで, 唯一つの軌道からなるリストが得られた. 代表元を拾って, 集合 set $= A \in \mathcal{M}$ を定義する.

```
gap> repset := List(list, c -> Representative(c));
[ [ (), (1,2)(3,4), (1,3)(2,4), (1,4)(2,3) ] ]
gap> set := repset[1];
[ (), (1,2)(3,4), (1,3)(2,4), (1,4)(2,3) ]
```

この集合 A の固定化群 $St_G(A)$ が G の 2-シロー群を与えるはずである.

```
gap> syl := Stabilizer(a4, set, OnSetsRight);
Group([ (1,3)(2,4), (1,2)(3,4) ])
gap> Size(syl);
```

となり, めでたく $G = A_4$ の 2-シロー群 syl が見つかった.

しかし, これが見つかったからといって何も感動しない. たかが A_4 の位数 4 の部分群くらい, こんな大がかりなことをしなくてもすぐ見つけられるではないか. せめて S_4 位やってもらわなくては, ということになるが, 実は S_4 で既に暗雲が立ち込めるのである. この場合, $|S_4| = 24$, S_4 の 2-シロー群の位数は 8 である. A_4 の場合と同様に, dom1 を 8 個の元からなる S_4 の部分集合の全体とする. A_4 の場合, dom の個数は $\binom{12}{4}$ だったから

```
gap> Binomial(12,4);
495
```

で与えられた. まあこれは許せる数である. S_4 の場合, dom1 の個数は $\binom{24}{8}$ なので

```
gap> Binomial(24,8);
735471
```

と飛躍的に大きくなる. dom1 は何とか定義できるが (735471 個の元のリスト), これを G-軌道に分解することは難しくなる. OrbitLengthsDomain の計算でさえも 30 分位待ってみたが答は戻って来なかった. まして, A_8 の 2-シロー群を計算しようと思えば, $\binom{|A_8|}{64}$ 個の元を持つリストを作らなければならない.

```
gap> Binomial(Order(a8),64);
219029071431847652079258104630491228176070783962544768926293 80
763409215634769314920372821317527795124407279582978130709946 29
378428372308625480638077548649564899778301834911702590290028375
```

リストを作るためには, それに見合う番地をコンピュータのメモリーに確保しなければならない. いかに GAP が大きな数を扱えるとはいえ, この巨大な数を前にしては, 一切の努力は無に帰して木枯しが吹きすさぶのみである. 嗚呼!! そういった事情で, この方法で 2-シロー群が見つけられる限界が, $G = A_4$ だったのである. 存在が保証されていることと, それを実際に見つけることがいかに違うかというのが北風の中で骨身にしみて分かるだろう. あたかも人工衛星から送られて来る地球全体の衛星写真を調べて迷子になった子どもを探すようなものである. しかし迷子を探すなら, 定廻りの旦那に相談するとか, 神田三河町の親分に聞いてみたほうがよっぽど早道であろう. 銀河系を縦断するような巨大リストを一挙に G-軌道に分解するといった荒仕事を企てるよりは, 足にまかせて, せっせと鉄瓶長屋を尋ね歩く方

が, 定めてあたりも見込まれるとしたものである. (蛇足ですが、パリにも鉄瓶横町 (rue de pot de fer) があります. 日本と姉妹横町になっているかどうかは知りません.) 我々も地道に, こつこつと, A_8 の 2-シロー群を見つけ出す方策を練ろう. まず次の性質に注目する.

(5.11.2) どんな p-群 H もその中心 $Z(H)$ は単位元以外の元を含む.

これを示そう. まず H を共役類に分解する. $H = \coprod_x C_x$ より $|H| = \sum_x |C_x|$. 共役類の元の個数 $|C_x|$ は $|H|$ の約数なので p の巾になる. したがって, $|C_x|$ が p で割り切れるかまたは $|C_x| = p^0 = 1$. $x = e$ の時, $|C_x| = 1$. $|H|$ は p で割れるので, $|C_x| = 1$ となる共役類 C_x は 2 個以上ある. $|C_x| = 1$ と $x \in Z(H)$ は同値なので, これより (5.11.2) が得られる.

以下, p-シロー群の存在を仮定して議論する. P を G の p-シロー群とし, $|P| = p^k$, とする. (5.11.2) より $Z = Z(P)$ は $\{1\}$ でない P の部分群である. そこで $|Z(P)|$ は p で割り切れる. $Z(P)$ はアーベル群であるから, ある $x \in Z(P)$ で x の位数が p になるものが存在する. $H = Z_G(x)$ とおくと, $P \subseteq H$. 今 $\langle x \rangle$ を x で生成される位数 p の群とすると, $\langle x \rangle$ は, P および H の正規部分群になる. $G_1 = H/\langle x \rangle$, $P_1 = P/\langle x \rangle$ とおく. P_1 は G_1 の p-シロー群になり, $|P_1| = p^{k-1}$ となる. $f : H \to G_1$ を自然な準同型とすると, $P = f^{-1}(P_1)$. この議論は次を示している.

(5.11.3) G には位数 p の元 x で $H = Z_G(x)$ の p-シロー群が G の p-シロー群になるものが存在する. $f : H \to G_1 = H/\langle x \rangle$ を自然な準同型とする. このとき, G_1 の p-シロー群 P_1 に対し, $P = f^{-1}(P_1)$ は G の p-シロー群を与える.

上の x の存在は p-シロー群の存在によって保証されるのである. (5.11.3) を G_1 に適用し, $x_1 \in G_1$ を見つけ, $H_1 = Z_{G_1}(x_1)$ を作る. さらに $f_1 : H_1 \to G_2 = H_1/\langle x_1 \rangle$ を作り, といったプロセスを繰り返すことにより, G の p-シロー群をみつける問題は, より小さい群 $G_1, G_2, \ldots$ での問題に帰着される. 実際, G_k の p-シロー群 P_k は $|P_k| = p^0 = 1$ であるから単位群である. この P_k を準同型写像 $f_{k-1} : H_{k-1} \to G_k$ 等で順次引き戻すことにより G の $p-$ シロー群 P が得られる. より精密に言えば, $|P_i| = p^{k-i}$ となる p-群 P_i と, $\mathrm{Ker}\, f_i \subset Z(P_i)$ をみたす全射準同型の列 $f_i : P_{i-1} \to P_i$ が得られたことになる.

(5.11.4) $$P \xrightarrow{f} P_1 \xrightarrow{f_1} P_2 \longrightarrow \cdots \longrightarrow P_{k-1} \xrightarrow{f_{k-1}} P_k = \{1\}.$$

実際の計算では, 何回かのプロセスの後, H_i が p-群になることがある. そうすれば $P_i = H_i$ が G_i の p-シロー群を与えるので, その先の計算は省略できる.

$G = A_8$ の場合にこの方法を実行してみよう. まず何回も使うので, 与えられた群 gp $= G$ に対して, 位数 p の元 x で $Z_G(x)$ の p-シロー群が G の p-シロー群に一致するものを求める関数 pelement(gp) を作っておく. 最後の条件は, 共役類 C_x の元の個数 $|C_x|$ が p と素になることと同値である.

```
gap> pelement := function(gp, p)
>          local class, list;
>          class := ConjugacyClasses(gp);
>          list := Filtered(class, c -> Order(Representative(c)) = p);
>          for c in list do
>             if Size(c) mod p  > 0 then
>                  return Representative(c);
```

```
>             fi;
>          od;
> end;
function( gp ) ... end
```

$G = S_4$ で確かめてみると

```
gap> pelement(s4,2);
(1,2)(3,4)
gap> Centralizer(s4, (1,2)*(3,4));
Group([ (1,2), (1,3)(2,4), (3,4) ])
gap> Size(last);
8
```

確かに $x = (1,2)(3,4)$ は位数 2 の元で, $|Z_G(x)| = 8$ は, 2-シロー群の位数 8 で割り切れる (この場合は既に 2-シロー群になっている). ついでに, $G = A_8$, $p = 3$ でやってみる.

```
gap> Factors(Order(a8));
[ 2, 2, 2, 2, 2, 2, 3, 3, 5, 7 ]
```

であるから, A_8 の 3-シロー群の位数は $3^2 = 9$ である.

```
gap> pelement(a8, 3);
(1,2,3)
gap> Centralizer(a8, (1,2,3));
Group([ (6,7,8), (5,6)(7,8), (4,5)(7,8), (1,2,3) ])
gap> Size(last);
180
```

さて話をもとに戻して $G = A_8$ についての計算を始めよう. まず $x \in G$ を定め, $H = Z_G(x)$ を構成する.

```
gap> x := pelement(a8,2);
(1,2)(3,4)(5,6)(7,8)
gap> H := Centralizer(a8,x);;
gap> Size(H);
192
```

$|A_8| = 20160$ だから, この時点で H は既に圧倒的に小さくなっている. 次に準同型 $f : H \to H/\langle x \rangle$ を定義する.

```
gap> C := Group(x);
Group([ (1,2)(3,4)(5,6)(7,8) ])
gap> f := NaturalHomomorphismByNormalSubgroup(H,C);
[ (5,6)(7,8), (5,7)(6,8), (3,4)(7,8), (3,5)(4,6), (1,2)(7,8), (1,3)(2,4) ] ->
[ f6, f1*f2^2, f5, f1*f2, f5*f6, f1*f2^2*f4 ]
```

```
gap> G_1 := Image(f);
Group([ f1, f2, f3, f4, f5, f6 ])
```

ここから第 2 段階に入る. G_1 の中に x_1 をみつけ, $H_1 = Z_{G_1}(x_1)$ を計算する.

```
gap> x_1 := pelement(G_1,2);
f3
gap> H_1 := Centralizer(G_1, x_1);
Group([ f1, f3, f4, f5, f6 ])
gap> Size(H_1);
32
```

$32 = 2^5$ だから, 幸運なことに H_1 はそれ自身既に 2 シロー群になっている. $H_1 = P_1$ とおく. $P = f^{-1}(P_1)$ として G の 2-シロー群が得られる.

```
gap> mysyl := PreImages(f, H_1);
Group([ (1,2)(3,4)(5,6)(7,8), (3,7)(4,8), (1,5)(2,6)(3,7)(4,8),
(1,3)(2,4)(5,7)(6,8), (3,4)(7,8), (5,6)(7,8) ])
gap> Size(mysyl);
64
```

これで, A_4 の 2-シロー群 P =mysyl が得られた. 一方, 5.7 節で A_4 の 2-シロー群 syl2 が定義されていた. 我々の作った mysyl と比べてみると

```
gap> syl2 = mysyl;
false
gap> IsConjugate(a8, syl2, mysyl);
true
```

等しくはないけれど, 2 つの 2-シロー群は (当然のことながら) 共役になっていることが確かめられる. 最初の方法では, S_4 ですでに沈没寸前だったが (S_4 の場合をコンピュータで何時間もかけて計算するのでは笑い話になってしまう), この方法なら共役類が計算できる限りは可能だろう. まだまだ余裕がありそうである. しかし上で確かめたようにそれは GAP の方法とは異なっている. GAP がどのような方法を使っているかは私の感知するところではない.

5.12 ブロックと置換表現

5.8 節の A_8 の共役類についてもう一度考える. 今 class を 112 個の元からなる A_8 の共役類とする. それは, 共役類のリスト ccl の 4 番目にあるから,

```
gap> class := ccl[4];
(1,2,3)^G
gap> Size(class);
112
```

$X =$ `class` は 112 個の点を持つ集合であり, $G = A_8$ が X に推移的に作用する. G は 112 個の点の置換を引き起こすので準同形 $\varphi : G \to S_{112}$ が定義される. 一般にこのような群 G から対称群 S_n への準同形を G の**置換表現**といい, n を表現の次数という. そこで, $G = A_8$ の 112 次の置換表現が得られたことになる. 交代群 A_8 は単純群なので, φ は単射準同形である. $G \simeq \mathrm{Im}\,\varphi$ は A_8 の S_{112} への埋め込みを与える. GAP では `op` $= \mathrm{Im}\,\varphi$ は次のようにして得られる.

```
gap> op := Action(a8, AsList( class ));
<permutation group with 6 generators>
gap> Size(op);
20160
```

S_{112} は巨大な群なので, `op` の生成元を具体的に書くだけでも大変なことになる. しかし群はちゃんと確定している. ここで, 前のように直接 `Action(norm, elab)` と書く代わりに `AsList(class)` によって共役類 `class` をその元からなるリストに置き換えた後, `Action` を実行していることに注意する. それは, GAP で共役類を確定する方法は必ずしも全ての元を決めるものではなく, 集合 (リスト) として与えられた方が共役類のままよりも計算が早くなるからである. しかし, 群が大きくなればなるほど, 集合として全てを確定するのは無理になるというのも極めて当然な話なのであって, そうした意味からも, 集合として表すといった直接的な方法を GAP は避けようと努力しているのである.

有限群 G が集合 X に推移的に作用しているとする. $|X| = n$ とすると, この作用により G の n 次の置換表現が得られる. X の部分集合 Δ が**ブロック**とは, 任意の $g \in G$ に対して

$$\Delta^g = \Delta \quad \text{または} \quad \Delta^g \cap \Delta = \emptyset$$

が成り立つことをいう. X に $1 < |\Delta| < |X|$ となるブロック Δ (自明でないブロック) が存在するとき, G は非原始的な置換群であるといい, そのようなブロックが存在しないとき, 原始的であるという. Δ を G のブロックとすると, Δ^g もまたブロックであり, $\{\Delta^g \mid g \in G\}$ によって X は共通部分のないいくつかのブロックの和集合に分割される (X の分割を与えるブロックの集合をブロック系という).

$$X = \coprod_{g \in \mathrm{St}_G(\Delta)\backslash G} \Delta^g \qquad \text{(disjoint union)} \tag{5.3.1}$$

ただし $\mathrm{St}_G(\Delta) = \{g \in G \mid \Delta^g = \Delta\}$ は Δ の固定化群 (集合 Δ を不変にする元 $g \in G$ の全体) である. そこで各ブロックを点とみなして集合 $Y = \{\Delta^g \mid g \in \mathrm{St}_G(\Delta)\backslash G\}$ を考えると G は Y の上に再び推移的に作用する. G が非原始的ならば $|Y| < |X|$ なので, より次数の低い m 次の置換表現が得られる ($m = |Y|$).

$x \in X$ の固定化群 $\mathrm{St}_G(x)$ を H とおくと, G の X への作用が推移的なことから, X は (G の作用込みで) 右剰余類の集合 $H\backslash G$ と同一視できる. 一方, Δ をブロックとすると $x \in \Delta$ に対して $\mathrm{St}_G(x) \subset \mathrm{St}_G(\Delta)$ が成り立つ. (実際, $g \in \mathrm{St}_G(x)$ ならば, $x \in \Delta \cap \Delta^g$ より $\Delta = \Delta^g$). Y は右剰余類の集合 $\mathrm{St}_G(\Delta)\backslash G$ と同一視される. このことから, G が原始的であることと, 一点の固定化群 $\mathrm{St}_G(x) = \{g \in G \mid x^g = x\}$ が G の極大部分群であることは同値になる. (H が G の極大部分群とは, H を含む G の部分群が G と H 以外に存在しないことをいう.)

`a8` の $X =$`class` への置換表現 `op` に戻ろう. GAP は 置換表現が原始的かどうかを判断してくれる. しかしそのためには `op` の作用域 $\{1, 2, \ldots, 112\}$ をはっきりと指定する必要がある. 例えば群

Group((2,3,4)) は集合 [1..4] $= \{1,2,3,4\}$ にも作用しているし, {2..4] にも作用している. op を集合 [1..113] の上への作用とみれば, 点 113 は a8 の作用の固定点になってしまう. この作用は推移的にならずブロックの議論は適用できない. そこで GAP では次のように書く.

```
gap> IsPrimitive(op, [1 .. 112]);
false
```

後半の [1 .. 112] が作用 op を 112 個の点の集合への置換とみなした上で, 原始的かどうかを問うているのである. 答は「非原始的」ということなので, 集合 class には自明でないブロックが存在する. GAP はそのようなブロックの中で個数が最小のブロックを選び, 集合 class を (5.3.1) のように分解する.

```
gap> blocks := Blocks( op, [1 .. 112]);
 [ [ 1, 7 ], [ 8, 14 ], [ 15, 21 ], [ 10, 26 ], [ 24, 40 ],
   [ 2, 13 ], [ 3, 19 ], [ 49, 54 ], [ 5, 31 ], [ 4, 25 ],
   [ 16, 27 ], [ 22, 28 ], [ 57, 70 ], [ 12, 38 ], [ 36, 42 ],
   [ 43, 48 ], [ 6, 37 ], [ 44, 53 ], [ 9, 20 ], [ 17, 33 ],
   [ 73, 77 ], [ 55, 60 ], [ 51, 64 ], [ 47, 68 ], [ 46, 63 ],
   [ 11, 32 ], [ 29, 35 ], [ 45, 58 ], [ 18, 39 ], [ 50, 59 ],
   [ 80, 90 ], [ 76, 89 ], [ 106, 109 ], [ 101, 104 ], [ 84, 91 ],
   [ 67, 72 ], [ 30, 41 ], [ 78, 82 ], [ 75, 85 ], [ 74, 81 ],
   [ 93, 96 ], [ 97, 100 ], [ 61, 66 ], [ 95, 102 ], [ 94, 99 ],
   [ 83, 87 ], [ 108, 110 ], [ 23, 34 ], [ 52, 69 ], [ 88, 92 ],
   [ 111, 112 ], [ 98, 103 ], [ 62, 71 ], [ 56, 65 ],
   [ 105, 107 ], [ 79, 86 ] ]
```

blocks は class のブロック系を与えるリストである. リストの各オブジェクトが 2 個の元からなるブロックになっている. 例えば, blocks[1] = [1, 7] がひとつのブロック Δ を与える. 他のブロック [8,14], [15, 21] などは, すべて Δ^g $(g \in G)$ により得られる. GAP に直接聞いてみれば

```
gap> Length(blocks[1]); Length(blocks);
2
56
```

これは, class が 2 個の元からなるブロック 56 個の和に分割されたことを意味する. そこで先に述べたように群 op をブロックの集合 Y (= blocks) に作用させることにより, 新しい置換表現 $\varphi_1 : \mathtt{op} \to S_{56}$ が得られる. $\mathrm{Im}\,\varphi_1$ を op1 とおくと, 交代群 $G = A_8$ の S_{56} への埋め込み a8 $\simeq$ op1 $\subset S_{56}$ が得られる.

```
gap> op1 := Action(op, blocks, OnSets);
<permutation group with 6 generators>
```

ここで注意することは, リスト blocks を構成しているオブジェクト (各ブロック) は集合であるから, それらのブロックを点とみなして群 op を作用させなければならない. OnSets は「集合の集まりの上に群を作用させなさい」という命令である. 新しく得られた置換群 op1 は原始的になる.

```
gap> IsPrimitive( op1, [1 .. 56]);
true
```

そこで先の議論により, op1 における一点の固定化群 M_1 は極大部分群になる. op1 は交代群 A_8 と同型であるからこれにより A_8 の極大部分群 M が得られることになる. しかし今 op1 は S_{56} の部分群として構成しているので, A_8 の極大部分群を具体的に書くためには同型 $\varphi_2 : A_8 \tilde{\to}$ op1 を決定し, $M = \varphi_2^{-1}(M_1)$ を求めなければならない. $G = A_8, H =$ op $\subset S_{112}$, $H_1 =$ op1 $\subset S_{56}$ とおく. 構成の仕方を思い出せば,

$$\varphi_2 : G \xrightarrow{\varphi} H \xrightarrow{\varphi_1} H_1$$

が同型 $\varphi_2 = \varphi \circ \varphi_1 : G \simeq H_1$ を与える (右作用のため写像の合成の順番が逆になっていることに注意.)

以上のプロセスは GAP では次のように実行される.

```
gap> ophom := ActionHomomorphism(a8, op);
<action homomorphism>
gap> ophom1 := ActionHomomorphism(op, op1);
<action homomorphism>
```

命令 ActionHomomorphism により a8 の置換表現 $\varphi =$ophom, op の置換表現 $\varphi_1 =$ophom1 が定義される. $\varphi_2 = \varphi \circ \varphi_1$ であるから

```
gap> composition := ophom*ophom1;;
```

により, $\varphi_2 =$composition が定義できた. H_1 の点 $x = 2$ での固定化群 $M_1 = \mathrm{St}_G(x)$ を stab とすると

```
gap> stab := Stabilizer(op2, 2);
<permutation group of size 360 with 5 generators>
```

によって H_1 の位数 360 の極大部分群 M_1 が定まる. $M = \varphi_2^{-1}(M_1)$ であるから

```
gap> preim := PreImages( composition, stab);
Group([ (2,5,7), (1,4)(2,7), (2,6,7), (1,3)(5,7), (6,8,7) ])
```

により, A_8 の極大部分群 $M =$preim が確定した. A_8 の位数が 20160 であることを考えると, 位数 360 の群 M が極大部分群になるというのは, 驚くべきことである. M は A_8 に比べてはるかに小さい群でありながら, それを含む A_8 の部分群は存在しないのである. これも A_8 が単純群であることの反映であろうか. (蛇足ながら, 単純群の条件は正規部分群が存在しないことであって, 単なる部分群については何も言っていない. しかし単純群においては部分群の存在も非常に制限されることを上の事実は示唆している.)

共役類 class から代表元 c1 を選び, c1 で生成される巡回群を xx とおく. a8 における xx の正規化群として得られる a8 の部分群を $K =$ sgp とする. K も位数 360 の部分群であり, 実は M と共役になる, 即ち $g^{-1}Kg = M$ となる $g \in G$ が存在する.

```
gap> c1 := Representative(class);
(1,2,3)
gap> xx := Group((c1));
Group([ (1,2,3) ])
gap> sgp := Normalizer(a8, xx);
Group([ (1,2,3), (5,6,7), (5,8)(6,7), (4,5,7,6,8), (2,3)(5,7,6,8) ])
gap> Size(sgp);
360
gap> RepresentativeAction(a8, sgp, preim);
(2,4)(7,8)
```

`RepresentativeAction(a8, sgp, preim)` が共役を与える元 $g \in G$ を見つける命令である. これより `g = (2,4)(7,8)` が得られる. 実際, K を共役により移してみると

```
gap> sgp^(2,4)(7,8) = preim;
true
```

により, $K^g = M$ が確かめられる.

6 章　ルービック・キューブ

6.1 ルービック・キューブ群の導入

ルービック・キューブは下図のように立方体を各面を 9 等分してできる 26 個の小立方体（キューブと呼ぶ）を移動させて, 各面の色を合わせて行くゲームである. 今, 立方体の各面は, 緑 (Green), 赤 (Red), 青 (Blue), 黄色 (Yellow), ピンク (Pink), 白 (White) に色分けされているとする. 以後, 色を表すのに, G, R, B, Y, W, P を使うことにする. また, 座標を固定するために, ルービック・キューブは常に図のように置かれているとする. ただし

$$\text{前面} \Leftrightarrow \text{G}, \quad \text{上面} \Leftrightarrow \text{R}, \quad \text{右面} \Leftrightarrow \text{B}, \quad \text{左面} \Leftrightarrow \text{W}, \quad \text{下面} \Leftrightarrow \text{P}, \quad \text{背面} \Leftrightarrow \text{Y}$$

である. ルービック・キューブの変換は各面の中心にあるキューブを動か さないとして良いので (実際にはこの位置での回転が入るが, 一色になって いる限り区別できない. 図のように文字を使うと, 実はこの文字は回転する) 以後は中心にあるキューブによってその面の色を固定して考える. そこで実質的に動くのは 20 個のキューブである. その内 8 個は立方体の頂点にあり, 残りの 12 個は立方体の辺の上に乗っている. これらをそれぞれ頂点キューブ, 辺キューブと呼ぶことにする.

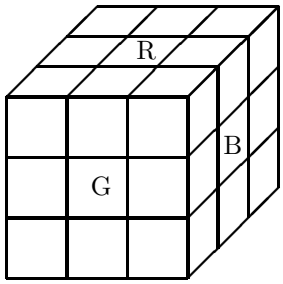


FIGURE 1. ルービック・キューブ $(3 \times 3 \times 3)$

ルービック・キューブの動きは, 軸を中心とする各面での 90 度の回転である. それらを, $x, x', y, y'z, z'$ と記すことにする. x は x 軸を中心とする G 面での回転, x' は x 軸を中心とする裏面 (Y 面）での回転 (以下同様）を表す. x, x', y, y', z, z' を繰り返して得られる変換をルービック変換と云う. ルービック変換は 20 個のキューブの回転付き置換とみることもできる. しかしここでは, 各面上の 8 個の正方形、合計 48 個に下図のように番号を付け, $x, \ldots, z'$ をこれらの 48 文字の置換とみなす. 置換 $x, x', y, y'z, z'$ で生成された S_{48} の部分群を**ルービック・キューブ群** ということにする.

`x_1, x_2, y_1, y_2, z_1, z_2` を G, Y, B, W, R, P をそれぞれ中心とする左回りの 90 度回転とする.

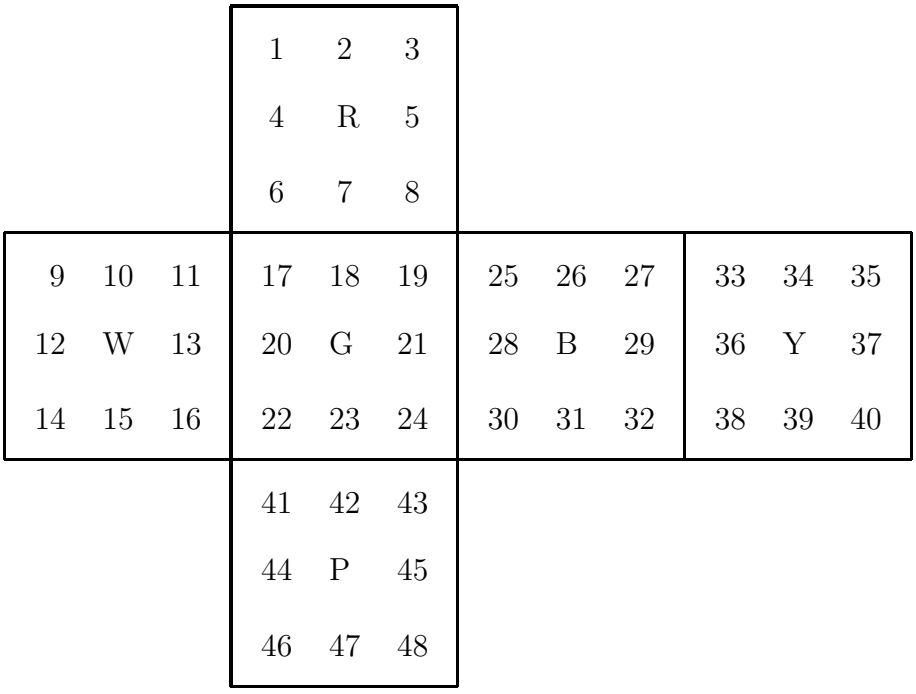


FIGURE 2. ルービック・キューブの展開図

```
gap> x_1 := (17, 19, 24, 22)(18, 21, 23, 20)(6, 25, 43, 16)
            (7, 28, 42, 13)(8, 30, 41, 11);
gap> x_2 := (33, 35, 40, 38)(34, 37, 39, 36)(3, 9, 46, 32)
            (2, 12, 47, 29)(1, 14, 48, 27);
gap> y_1 := (25, 27, 32, 30)(26, 29, 31, 28)(3, 38, 43, 19)
            (5, 36, 45, 21)(8, 33, 48, 24);
gap> y_2 := (9, 11, 16, 14)(10, 13, 15, 12)(1, 17, 41, 40)
            (4, 20, 44, 37)(6, 22, 46, 35);
gap> z_1 := (1, 3, 8, 6)(2, 5, 7, 4)(9, 33, 25, 17)
            (10, 34, 26, 18)(11, 35, 27, 19);
gap> z_2 := (41, 43, 48,46)(42, 45, 47, 44)(14, 22, 30, 38)
            (15, 23, 31, 39)(16, 24, 32, 40);
gap> cube := Group(x_1, x_2, y_1, y_2, z_1, z_2);
gap> Size(cube);
     43252003274489856000
```

これによりルービック・キューブ群 H =cube が位数 43252003274489856000 の S_{48} の部分群として確定する. この巨大な位数を因数分解すると,

```
gap> Collected(Factors(last));
[ [ 2, 27 ], [ 3, 14 ], [ 5, 3 ], [ 7, 2 ], [ 11, 1 ] ]
```

を得る. すなわち, $|H| = 2^{27} \cdot 3^{14} \cdot 5^3 \cdot 7^2 \cdot 11$.

以下, 群 H =cube の 48 点への作用 (置換表現) を詳しく調べていく. まず軌道分解を求めると

```
gap> orbits := Orbits(cube, [1..48]);
   [ [ 1, 14, 17, 3, 48, 9, 22, 19, 41, 38, 8, 27, 24, 46,
       11, 33, 30, 40, 43, 6, 32, 35, 16, 25 ],
     [ 2, 12, 5, 47, 10, 36, 7, 29, 44, 13, 34, 45, 28, 4,
       31, 37, 42, 15, 26, 21, 20, 39, 23, 18 ] ]
gap> List(orbits, c -> Size(c));
   [ 24, 24 ]
```

と 2 個の軌道に分解する. 数字を比べて見れば分かるように 1 番目の軌道が頂点キューブの置換から得られる軌道で, 2 番目の軌道が辺キューブの置換から得られる軌道である. 以後, これらを**頂点軌道**, **辺軌道**ということにする. ルービック・キューブ群 cube の群構造を決定するために cube の頂点軌道への作用と辺軌道への作用をそれぞれ個別に調べる.

6.2 頂点ルービック・キューブ群

まず, cube の頂点軌道 orbits[1] への置換表現を調べる. $f_V : H \to S_{24}$ を H =cube の orbits[1] への置換表現とし, $H_V = \mathrm{Im}(f_V)$ =cube1, f_V =hom1 とおく. H_v を**頂点ルービック・キューブ群**という.

```
gap> cube1 := Action(cube, orbits[1]);
<permutation group with 6 generators>
gap> Size(cube1);
88179840
gap> hom1 := ActionHomomorphism(cube, orbits[1]);
<action homomorphism>
```

これにより, 置換群 H_V = cube1 $\subset S_{24}$ と全射準同型 f_V = hom1 : $H \to H_V$ が確定する. orbits[1] は H 軌道のひとつだから当然 H_V の作用は推移的である. でも, 念のため確かめてみよう. 人生何事も慎重にやるに越したことはない. まず cube から始めると,

```
gap> IsTransitive(cube, orbits[1]);
true
```

となり何の問題もない. そこで調子の乗って cube1 に移ると

```
gap> IsTransitive(cube1, orbits[1]);
false
```

となって, ヤヤヤ !!ということになる. 大人物はそういう時「コンピュータだってたまには間違うこともあるさ」と泰然自若としていられるのであるが, 我々小人は呆然自失してとまどうばかり. 実は cube1 の作用域は既に [1..24] になっている. orbits[1] のままでは, 作用域は [1..48] の部分集合に過ぎないので推移的にはならないのである. そこで反省し心を入れ換えて,

```
gap> IsTransitive(cube1, [1..24]);
true
```

とすると, cube1 の作用が推移的であることが確かめられる. めでたし, めでたし.

元のお話に戻って cube1 の [1..24] への推移的な作用は果して原始的かどうかを考えよう.

```
gap> corners := Blocks(cube1, [1..24]);
[ [ 1, 6, 22 ], [ 2, 14, 18 ], [ 3, 15, 20 ], [ 4, 12, 16 ],
  [ 5, 10, 21 ], [ 7, 9, 23 ], [ 8, 11, 24 ], [ 13, 17, 19 ] ]
```

従って, cube1 は原始的ではなく, 最小ブロックは 3 個の元からなる. orbits[1] と [1..24] との対応表

1	14	17	3	48	9	22	19	41	38	8	27	24	46	11	33	30	40	43	6	32	35	16	25
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

により, block [1, 6, 22] は元の番号付け [1, 9, 35] に対応し, [2, 14, 18] は [14, 46, 40] に対応する. 以下同様にして, 8 個のブロックが立方体の 8 個の頂点に対応することが分かる (図 4 参照). 従ってこれらのブロックの置換は立方体の頂点の置換に対応し, 各ブロックを動かさない cube1 の元は対応する頂点における頂点キューブの回転に対応する.

次の図のように, $2\times2\times2$ 個のキューブからなる立体を考える. これはミニキューブとかポケットキューブとか呼ばれているルービック・キューブの変形版である. 上に述べたように H_V =cube1 を考えることはもとのルービック・キューブの代わりに, 頂点キューブのみからなるミニキューブを扱うことになる. H_V での生成元 $a_1, a_2, b_1, b_2, c_1, c_2$ が, それぞれ前面, 背面, 右面, 左面, 上面, 背面 での 90 度回転に対応する.

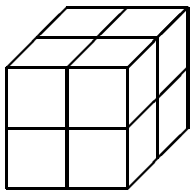


FIGURE 3. ミニキューブ $(2\times2\times2)$

orbits[1] と [1..24] の対応のもとにミニキューブの展開図を以下に載せておく. 以前に与えた生成元 $a_1, a_2 \ldots, c_2$ の表示はここに載せた番号付けのもとでのミニキューブの動作に一致している.

H_V =cube1 のブロックの集合への置換表現 $\varphi_V : H_V \to S_8$ の像を $\mathrm{Im}\,\varphi_V$ =cube1b とおく.

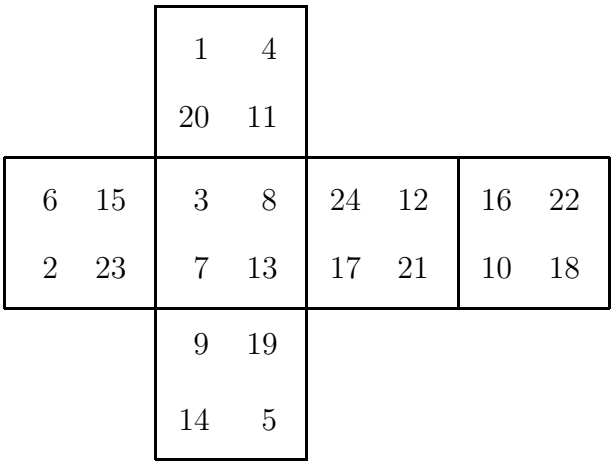


FIGURE 4. ミニキューブの展開図

```
gap> cube1b := Action(cube1, corners, OnSets);
Group([ (3,7,8,6), (1,2,5,4), (4,5,8,7), (1,3,6,2), (1,4,7,3), (2,6,8,5) ])
gap> Size(cube1b);
40320
```

cube1b は S_8 の部分群である. 一方, GAP の階乗関数を使って,

```
gap> Factorial(8);
40320
```

従って, S_8 の位数は 40320 で, それは cube1b の位数に一致する. $H_V \simeq S_8$ が成立し，全射準同型 $\varphi_V : H_V \to S_8$ が得らえた. 次に $\mathrm{Ker}\,\varphi_V$ について調べよう. まず写像 φ_V =blockhom1 を GAP に読み込む.

```
gap> blockhom1 := ActionHomomorphism(cube1, cube1b);
<action homomorphism>
```

$K_V = \mathrm{Ker}\,\varphi_V$ =ker1 について調べてみよう.

```
gap> ker1 := Kernel(blockhom1);
<permutation group with 7 generators>
gap> Size(elab);
2187
gap> Factors(Size(ker1));
[ 3, 3, 3, 3, 3, 3, 3 ]
gap> IsElementaryAbelian(ker1);
true
```

これで, ker1 が位数 3^7 の基本アーベル群になることが分かった. すなわち $K_V = \mathrm{Ker}\,\varphi_V \simeq (\mathbb{Z}/3\mathbb{Z})^7$ が成り立つ. 以上の議論から完全系列

$$1 \longrightarrow (\mathbb{Z}/3\mathbb{Z})^7 \longrightarrow H_V \xrightarrow{\varphi_V} S_8 \longrightarrow 1 \tag{6.2.1}$$

が得られる.

K_V について更に詳しく調べる. 準同形 φ_V はブロックの上への置換として定義されたので, $K_V = \mathrm{Ker}\,\varphi_V$ は各ブロックを全て（集合として）保存する. それは, ミニキューブの各頂点を全て固定する変換である. ミニキューブでの動きを考慮するとそのような変換は各頂点での頂点キューブの回転に他ならないことが分る. 従って K_V は各頂点での回転から生成される群 $(\mathbb{Z}/3\mathbb{Z})^8$ に含まれる. $K_V \simeq (\mathbb{Z}/3\mathbb{Z})^7$ が $(\mathbb{Z}/3\mathbb{Z})^8$ のどのような部分群かを決定しよう. まず, 8 個の頂点を

```
[1,6,22], [4,12,16], [8,11,24], [3,15,20],
[7,9,23], [13,17,19], [5,10,21], [2,14,18]
```

の順番に並べる. これらはミニキューブにおいて辺で結ばれた頂点が隣り合うように並べたものである. i 番目の頂点での 120 度の左回転を ε_i と表す. $\varepsilon_1, \ldots, \varepsilon_8$ が $(\mathbb{Z}/3\mathbb{Z})^8$ の生成元を与える. ε_1 =(1,6,22), ε_2 =(4,16,12) に対して

```
gap> (1,6,22) in cube1;
false
gap> (1,6,22)(4,12,16) in cube1;
true
```

つまり, ε_1 は cube1 に含まれず, $\varepsilon_1\varepsilon_2^{-1}$ は cube1 に含まれることが分る. 言い替えれば, $\varepsilon_1\varepsilon_2^{-1} \in K_V$ が成り立つ. ミニキューブの対称性より $\varepsilon_2\varepsilon_3^{-1}, \ldots, \varepsilon_7\varepsilon_8^{-1}$ も K_V に含まれるはずである. 実際確かめてみると,

```
gap> (4,16,12)(11,8,24) in cube1;
true
gap> (11,24,8)(20,15,3) in cube1;
true
gap> (7,9,23)(3,20,15) in cube1;
true
gap> (13,17,19)(7,23,9) in cube1;
true
gap> (19,13,17)(5,10,21) in cube1;
true
gap> (5,21,10)(14,2,18) in cube1;
true
```

以上より, $\varepsilon_1\varepsilon_2^{-1}, \varepsilon_2\varepsilon_3^{-1}, \ldots, \varepsilon_7\varepsilon_8^{-1}$ が K_V の生成元を与え,

$$K_V = \left\{(\varepsilon_1^{\lambda_1}, \ldots, \varepsilon_8^{\lambda_8}) \in (\mathbb{Z}/3\mathbb{Z})^8 \mid \sum_{i=1}^{8} \lambda_i \equiv 0 \pmod 3\right\} \tag{6.2.2}$$

となることが確かめられた.

最後に完全系列 (6.2.1) が分裂すること, すなわち H_V が $(\mathbb{Z}/3\mathbb{Z})^7$ と S_8 の半直積 $S_8 \ltimes (\mathbb{Z}/3\mathbb{Z})^7$ に同型になることを示そう. もし半直積になるとすれば, H_V は S_8 と同型な部分群を含むはずであり, この部分群は頂点キューブの (回転なしの) 置換の全体として得られるはずである. そこで,8 個の頂点の集合 [1,2,3,4,5,7,8,13] の置換として得られる H_V =cube1 の部分群を comp1 とする. 言い替えると comp1 は, [1,2,3,4,5,7,8,13] の集合としての固定化群になっている. そこで

```
gap> comp1 := Stabilizer(cube1, [1,2,3,4,5,7,8,13], OnSets);
<permutation group of size 40320 with 8 generators>
```

により, S_8 と同じ位数を持つ cube1 の部分群 comp1 が定義される. $\varphi_V : H_V \to S_8$ の comp1 への制限は全射になる. 実際

```
gap> Action(comp1, corners, OnSets) = cube1b;
true
```

従って, φ_V の comp1 への制限は同型 comp1 $\simeq S_8$ を与え, これより (6.2.1) が分裂することが導かれる. 直接, 半直積になることを確かめるには, comp1 $\cap K_V = 1$ であることと, H_V が K_V と comp1 で生成されることを見れば良い.

```
gap> Size(Intersection(comp1, ker1));
1
gap> Closure(comp1, ker1) = cube1;
true
```

以上で, $H_V \simeq S_8 \ltimes (\mathbb{Z}/3\mathbb{Z})^7$ となることが確かめられた.

注意. 一般に対称群 S_n と巡回群 $\mathbb{Z}/r\mathbb{Z}$ とのレス積, すなわち S_n と $(\mathbb{Z}/r\mathbb{Z})^n$ との半直積 $S_n \ltimes (\mathbb{Z}/r\mathbb{Z})^n$ (S_n は $(\mathbb{Z}/r\mathbb{Z})^n$ に置換として作用) を $G(r,1,n)$ と表す (複素鏡映群としての記号). $(\mathbb{Z}/r\mathbb{Z})^n$ の生成元を $\varepsilon_1, \ldots, \varepsilon_n$ とするとき半直積 $S_n \ltimes E_r$ を $G(r,r,n)$ と表す. ただし E_r は

$$E_r = \left\{ (\varepsilon_1^{\lambda_1}, \ldots, \varepsilon_n^{\lambda}) \in (\mathbb{Z}/r\mathbb{Z})^n \mid \sum_{i=1}^{n} \lambda_i \equiv 0 \pmod{r} \right\} \simeq (\mathbb{Z}/r/ZZ)^{n-1}$$

で定義される $(\mathbb{Z}/r\mathbb{Z})^n$ の部分群である. $G(r,r,n)$ は位数 $n! \times r^{n-1}$ の複素鏡映群になる. 以上の記号のもとに, $H_V \simeq G(3,3,8)$ と表される.

6.3 辺ルービック・キューブ群

次に cube の辺軌道 orbits[2] への置換表現を調べる. $f_E : H \to S_{24}$ を H =cube の orbits[2] への置換表現とし, $H_E = \mathrm{Im}(f_E)$ =cube2, f_E =hom2 とおく. H_E を**辺ルービック・キューブ群**という.

```
gap> cube2 := Action(cube, orbits[2]);
<permutation group with 6 generators>
gap> Size(cube2);
980995276800
gap> hom2 := ActionHomomorphism(cube, orbits[2]);
<action homomorphism>
```

辺ルービック・キューブ群 H_E が位数 980995276800 の群として確定する. H_E =cube2 の oribts[2] への作用は推移的であるから, 前と同様にブロック分解ができる.

```
gap> edges := Blocks(cube2, [1..24]);
[ [ 1, 11 ], [ 2, 16 ], [ 3, 19 ], [ 4, 22 ], [ 5, 14 ],
  [ 6, 8 ], [ 7, 24 ],  [ 9, 18 ], [ 10, 21 ], [ 12, 15 ],
  [ 13, 20 ], [ 17, 23 ] ]
```

orbits[2] と [1..24] との対応表は次のようになる.

2	12	5	47	10	36	7	29	44	13	34	45	28	4	31	37	42	15	26	21	20	39	23	18
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

この対応表により, 次の辺キューブの展開図が得られる.

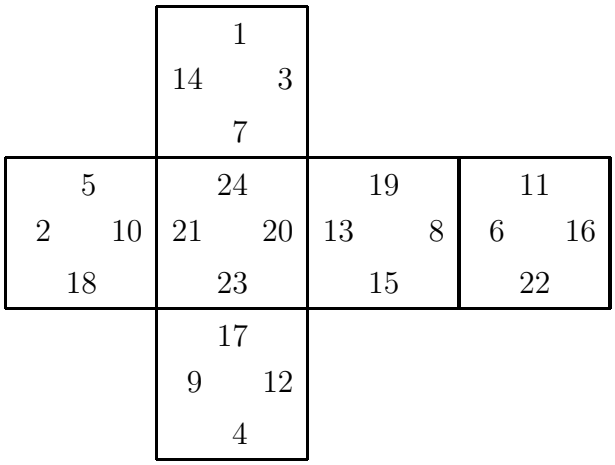


FIGURE 5. 辺キューブの展開図

これより, 各ブロックが 12 個の辺キューブに対応していることが分かる.

H_E =cube2 のブロックの集合への置換表現 $\varphi_E : H_E \to S_{12}$ の像を $\mathrm{Im}\,\varphi_E$ =cube2b とおく.

```
gap> cube2b := Action(cube2, edges, OnSets);
Group([ ( 7,11,12, 9), ( 1, 2, 4, 6), ( 3, 6,10,11),
        ( 2, 5, 9, 8), ( 1, 3, 7, 5), ( 4, 8,12,10) ])
gap> Size(cube2b);
479001600
```

cube2b は S_{12} の部分群であり, 前と同様に 12! を計算して

```
gap> Factorial(12);
479001600
```

cube2b $\simeq S_{12}$ を得る. 特に, $\varphi_E : H_E \to S_{12}$ は全射準同型になる. φ_E =blockhom2, $\mathrm{Ker}\,\varphi_E = K_V$ =ker2 とおく. K_E =ker2 を計算しよう.

```
gap> blockhom2 := ActionHomomorphism(cube2, cube2b);
<action homomorphism>
gap> ker2 := Kernel(blockhom2);
<permutation group with 11 generators>
gap> Size(ker2);
2048
gap> Factors(Size(ker2));
[ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
gap> IsElementaryAbelian(ker2);
true
```

以上により, ker2 が位数 2^{11} の基本アーベル群であることが分かった. $K_E \simeq (\mathbb{Z}/2\mathbb{Z})^{11}$ であるから, 完全系列

$$1 \longrightarrow (\mathbb{Z}/2\mathbb{Z})^{11} \longrightarrow H_E \xrightarrow{\varphi_E} S_{12} \longrightarrow 1 \tag{6.3.2}$$

が得られる.

$K_E \simeq (\mathbb{Z}/2\mathbb{Z})^{11}$ の構造を決めよう. 辺キューブの動きを見ることにより, K_E は $(\mathbb{Z}/2\mathbb{Z})^{12}$ に含まれる. ここに各成分 $\mathbb{Z}/2\mathbb{Z}$ は辺キューブの 180 度回転に対応する. 前と同様に 12 個のブロックに次のような順序を入れる.

```
[1,11], [3,19], [7,24], [5,14], [2,16], [9,18],
[10,21], [17,23], [13, 20], [12,15], [6,8], [4,22]
```

この順序で隣り合ったブロックに対応する辺キューブは立方体の中で最短距離を取るものになっている. つまり, 隣り合った辺キューブの位置関係はどれも同じである. i 番目のブロックに対応する辺ブロックの 180 度の回転を ε_i とおく. $\varepsilon_1, \ldots, \varepsilon_{12}$ が $(\mathbb{Z}/2\mathbb{Z})^{12}$ の生成元を与える. $\varepsilon_1 = (1, 11)$, $\varepsilon_2 = (3, 19)$ に対し

```
gap> (1,11) in cube2;
false
gap> (1,11)(3,19) in cube2;
true
```

従って, $\varepsilon_1 \notin K_E$, $\varepsilon_1\varepsilon_2 \in K_E$ である. $\varepsilon_1\varepsilon_2, \varepsilon_2\varepsilon_3, \ldots \varepsilon_{11}\varepsilon_{12}$ は全てルービック・キューブの中で対称な位置を占めることから, $\varepsilon_i\varepsilon_{i+1} \in K_E$ $(1 \leq i \leq 11)$ となることが分かる. これより,

$$K_E = \left\{(\varepsilon_1^{\lambda_1}, \ldots, \varepsilon_{12}^{\lambda_{12}}) \in (\mathbb{Z}/2\mathbb{Z})^{12} \mid \sum_{i=1}^{12} \lambda_i \equiv 0 \pmod 2\right\}$$

が得られる.

最後に, 完全系列 (6.3.2) が分裂すること, すなわち H_E が S_{12} と $(\mathbb{Z}/2\mathbb{Z})^{11}$ の半直積になることを示す.

```
gap> comp2 := Stabilizer(cube2, [1,2,3,4,5,6,7,9,10,12,13,17], OnSets);
<permutation group of size 479001600 with 8 generators>
gap> Action(comp2, edges, OnSets) = cube2b;
true
```

頂点キューブの場合と同様に, cube2 の中に S_{12} と同じ位数を持つ部分群 comp2 が定義され, $\varphi_E : H_E \to S_{12}$ の comp2 への制限は同型 comp2 $\simeq S_{12}$ を与える. 従って, $H_E \simeq S_{12} \ltimes (\mathbb{Z}/2\mathbb{Z})^{11}$ が得られる. 前節の記号に従えば, これは $H_E \simeq G(2, 2, 12)$ を意味する.

6.4 ルービック・キューブ群の決定

ルービック・キューブ群 H は 48 点の置換群であり, その 48 点がふたつの軌道 orbits[1] と orbits[2] に分解する. H から誘導された orbits[1] 上の置換群が H_V であり, orbits[2] 上の置換群が H_E である. $f_V : H \to H_V$, $f_E : H \to H_E$ を前節で定義した全射準同型とすると $f : H \to H_V \times H_E, g \mapsto (f_V(g), f_E(g))$ は単射準同型を与える. $\varphi_V : H_V \to S_8$, $\varphi_E : H_E \to S_{12}$ を前節のように取る. 次の図式を考える.

$$\widetilde{f} : H \xrightarrow{f} H_V \times H_E \xrightarrow{\varphi_V \times \varphi_E} S_8 \times S_{12} \tag{6.4.1}$$

S_8 は文字 $\{1, 2, \ldots, 8\}$, S_{12} を文字 $\{9, 10, \ldots, 21\}$ に関する置換群とみて $S_8 \times S_{12}$ を S_{21} の部分群とみなす. $A(S_8 \times S_{12}) = (S_8 \times S_{12}) \cap A_{21}$ とおく (A_{21} は 21 次の交代群). $A(S_8 \times S_{12})$ は $S_8 \times S_{12}$ の指数 2 の部分群になる. 次を示す.

$$\widetilde{f}(H) \subset A(S_8 \times S_{12}). \tag{6.4.2}$$

実際 $g = x_1, x_2, \ldots, z_2$ に対して, $\varphi_V \circ f_V(g)$ は位数 4 の巡回置換になる. これは $\varphi \circ f_V$ が回転を無視した頂点キューブの置換であることより当然の結果であるが, 直接確かめようとすれば

```
gap> hom11 := hom1*blockhom1;;
gap> Image(hom11, x_1); Image(hom11, x_2);
(3,7,8,6)
(1,2,5,4)
gap> Image(hom11, y_1); Image(hom11, y_2);
(4,5,8,7)
```

```
(1,3,6,2)
gap> Image(hom11, z_1); Image(hom11, z_2);
(1,4,7,3)
(2,6,8,5)
```

同様に, $\varphi_E \circ f_E(g)$ もまた位数 4 の巡回置換である.

```
gap> hom22 := hom2*blockhom2;;
gap> Image(hom22, x_1); Image(hom22, x_2);
(7,11,12,9)
(1,2,4,6)
gap> Image(hom22, y_1); Image(hom22, y_2);
(3,6,10,11)
(2,5,9,8)
gap> Image(hom22, z_1); Image(hom22, z_2);
(1,3,7,5)
(4,8,12,10)
```

上の結果から $g = x_1, x_2, \ldots, z_2$ に対して, $\widetilde{f}(g) = (\varphi_V \circ f_V(g), \varphi_E \circ f_E(g)) \in S_8 \times S_{12}$ は位数 4 の巡回置換の積である. 従って, $\widetilde{f}(g) \in A(S_8 \times S_{12})$. $x_1, x_2, \ldots, z_2$ は H の生成元であるから, これより (6.4.2) が得られる.

最後に

$$H \simeq f(H) = A(S_8 \times S_{12}) \ltimes ((\mathbb{Z}/3\mathbb{Z})^7 \times (\mathbb{Z}/2\mathbb{Z})^{11}) \tag{6.4.3}$$

を示そう. ルービック・キューブ群 H の構造は (6.4.3) により完全に記述される.

(6.4.2) より, $H \simeq f(H) \subset (\varphi_V \times \varphi_E)^{-1}(A(S_8 \times S_{12}))$. ここで

$$(\varphi_V \times \varphi_E)^{-1}(A(S_8 \times S_{12})) \simeq A(S_8 \times S_{12}) \ltimes ((\mathbb{Z}/3\mathbb{Z})^7 \times (\mathbb{Z}/2\mathbb{Z})^{11})$$

は $H_V \times H_E$ の指数 2 の部分群になる. そこで, $H =$cube の位数と $H_V \times H_E =$ cube1 $\times$ cube2 の位数の半分を比較して

```
gap> Size(cube) = Size(cube1)*Size(cube2)/2;
true
```

これより $f(H) = (\varphi_V \times \varphi_E)^{-1}(A(S_8 \times S_{12}))$ となり, (6.4.3) が成り立つ.

最後にルービック・キューブ群 H の中心を調べておこう.

```
gap> Center(cube);
Group([ ( 2,34)( 4,10)( 5,26)( 7,18)(12,37)(13,20)
        (15,44)(21,28)(23,42)(29,36)(31,45)(39,47) ])
```

すなわち, H の中心は $\{1, g\}$ からなる. ここで

$$g = (2,34)(4,10)(5,26)(7,18)(12,37)(13,20)(15,44)(21,28)(23,42)(29,36)(31,45)(39,47)$$

は, 12 個の辺キューブを全て 180 度ひっくり返す位数 2 の変換である.

6.5 生成元による表示

ルービック・キューブ群 H は $x_1, x_2, y_1, y_2, z_1, z_2$ から生成されていた. 与えられた H の元をこれらの生成元の積として表すことを考えてみよう. これがルービック・キューブの「バラバラになったパターンを如何にして元の状態に復元させるか」というゲームの主要テーマに他ならない.

まず頂点をそろえることから始める. H の生成元 $x_1, x_2, y_1, y_2, z_1, z_2$ の H_V での像をそれぞれ $a_1, a_2, b_1, b_2, c_1, c_2$ とおくと以下のようになる.

```
gap> a_1:= Image(hom1, x_1); a_2 := Image(hom1, x_2);
( 3, 8,13, 7)( 9,15,11,17)(19,23,20,24)
( 1, 2, 5,12)( 4, 6,14,21)(10,16,22,18)
gap> b_1 := Image(hom1, y_1); b_2 := Image(hom1, y_2);
( 4,10,19, 8)( 5,13,11,16)(12,21,17,24)
( 1, 3, 9,18)( 2, 6,15,23)( 7,14,22,20)
gap> c_1 := Image(hom1, z_1); c_2 := Image(hom1, z_2);
( 1, 4,11,20)( 3, 6,16,24)( 8,15,22,12)
( 2, 7,17,10)( 5,14, 9,19)(13,21,18,23)
```

H_V の元 $q = a_1 b_1 a_1^{-1}$, $r = c_1^{-1} b_1^{-1} c_1 b_1$ を考える.

```
gap> q := a_1*b_1*a_1^-1;
( 3, 4,10,24)( 5, 8,15,16)(11,20,12,21)
gap> r := c_1^-1*b_1^-1*c_1*b_1;
( 3,24,20,11,15, 8)( 4,10,16, 5,12,21)
```

$q, r \in H_V$ の $\varphi_V : H_V \to S_8$ への像を計算すると,

```
gap> qq := Image(blockhom1, q);
(3,4,5,7)
gap> rr := Image(blockhom1,r);
(3,7)(4,5)
gap> rr*qq;
(4,7)
```

特に rq の像は S_8 の互換を与える. つまり rq は回転を無視すれば互換になる. rq 自身を書いてみると

```
gap> r*q;
( 4,24,12,11,16, 8)( 5,21,10)
```

すなわち rq はブロック [4,12,16] と [8,11,24] に対応する頂点を交換し, ブロック [5,10,21] に対応する頂点の回転を引き起こす. Figure 4 を参照して, $rq = (c_1^{-1}b_1^{-1}c_1b_1)(a_1b_1a_1^{-1})$ がミニキューブの隣り合った 2 頂点の置換であることが分かる. ルービック・キューブの対称性から, 生成元 a_1, b_1, c_1 を適当に他の生成元と取り換えることにより任意の隣り合った 2 頂点の間の置換が同様にして得られる. 今 $g \in H_V$ を取る. $\varphi_V(g) \in S_8$ はこれらの互換の積で書けるので, g に右から rq 達を繰り返し掛けることにより $g' \in K_V = \mathtt{ker1}$ に変形することができる. (言い替えると, rq 達を繰り返し適用することにより, ミニキューブの各頂点を (回転を除いて) 正しい位置に持って来ることができる).

そこで $g \in K_V$ と仮定してよい. $\varphi_V(rq)$ は互換だから, $(rq)^2 \in K_V$ である.

```
gap> (r*q)^2 in ker1;
true
```

計算してみると,

```
gap> (r*q)^2;
( 4,12,16)( 5,10,21)( 8,24,11)
```

すなわち $(rq)^2$ は 3 頂点 [4,12,16], [5,10,21], [8,24,11] おける 120 度の右回転である. 一方, a_1, b_1, c_1 を c_1, b_1, a_2 に取り換えて,

```
gap> r_1 := a_2^-1*b_1^-1*a_2*b_1;
( 1,12,22,16, 6, 4)( 5,13,21,17,10,19)
gap> q_1 := c_1*b_1*c_1^-1;
( 1,10,19,12)( 4, 6, 5,13)(16,22,21,17)
gap> r_1*q_1;
( 4,10,12,21,16, 5)(13,17,19)
gap> (r_1*q_1)^2;
( 4,12,16)( 5,10,21)(13,19,17)
```

そこで, $(r_1q_1)^{-2}(rq)^2$ を計算すると

```
gap> (r_1*q_1)^-2*(r*q)^2;
( 8,24,11)(13,17,19)
```

これは, となりあった 2 頂点 [8,11,24], [13,17,19] での右 120 度, 左 120 度の回転である. 生成元を取り換えることにより, 全ての隣り合った 2 頂点で, このような変換を作ることができて, それらが, K_V を生成する. 従って $g \in K_V$ は, $(r_1q_1)^{-2}(rq)^2$ の形の元の積で書ける. 以上の操作で頂点は全て (回転も含めて) 正しい位置に持って行くことができる.

$p_V : H \to H_V$ を $H_V \times H_E$ から H_V への射影を H に制限したものとする. $\operatorname{Ker} p_V = H \cap H_E \simeq A_{12} \ltimes (\mathbb{Z}/2\mathbb{Z})^{11}$ である. 今までの議論から $g \in \operatorname{Ker} p_V$ (隅はすべてそろっている) としてよい. $u = (x_1^2y_1^2)^3$ とおく.

```
gap> u := (x_1^2*y_1^2)^3;
( 5,45)( 7,42)(18,23)(26,31)
gap> Image(hom2, u);
( 3,12)( 7,17)(15,19)(23,24)
```

従って, u は Figure 5 における 辺キューブ [7,24] と [17,23] を入れ換え, [3,19] と [12,15] を入れ換え, 他はすべて固定する変換である. $\varphi_E \circ f_E(u) \in S_{12}$ は互換 2 個の積になる. $\varphi_E \circ f_E : H \to S_{12}$ は全射なので, A_8 の全ての元を u の共役元いくつかの積 (の像) として構成できる. そこで g に u の共役元達を適当に掛けることにより, $\varphi_E \circ f_E(g) = 1$, すなわち $g \in (\mathbb{Z}/2\mathbb{Z})^{11}$ とできる. 回転を除いて辺の位置はを全てそろった状態になる.

最後に, $(\mathbb{Z}/2\mathbb{Z})^{11}$ の元を H の生成元を使って現わそう. $v_1 = y_1^{-1}(x_2 y_1 x_2^{-1} y_1^{-1})$ とおき, u の共役元 $w_1 = v_1^{-1} u v_1$ を考える.

```
gap> v_1 := y_1^-1*(x_2*y_1*x_2^-1*y_1^-1);
( 3,27,33)( 5,21,45)( 8,24,46,32,25,30,40,38,19,43,14,48)
(26,28,31)(29,47)(36,39)
gap> w_1 := v_1^-1*u*v_1;
( 5,21)( 7,42)(18,23)(26,28)
gap> Image(hom2, v_1);
( 3,20,12)( 4, 8)( 6,22)(13,15,19)
gap> Image(hom2, w_1);
( 3,20)( 7,17)(13,19)(23,24)
```

w_1 の f_E での像は, 辺キューブ [7,24] と [17,23] を入れ換え, [3,19] と [20,13] をこの順序で入れ換える. 一方, $v_2 = x_1^{-1} z_1^{-1} x_1 y_1^2 x_2^{-1} y_1^{-1}$ とおき, u の共役元 $w_2 = v_2^{-1} u v_2$ を考える.

```
gap> v_2 := x_1^-1*z_1^-1*x_1*y_1^2*x_2^-1*y_1^-1;
( 1,46, 9,40,35,14)( 2, 4,47,12)( 3,25,33,19,27, 8)( 5,26)(10,39,37,34)
(21,29,31,28,36,45)(24,32,30,38,43,48)
gap> Image(hom2, v_2);
( 1,14, 4, 2)( 3,19)( 5,22,16,11)( 6,12,20, 8,15,13)
gap> w_2 := v_2^-1*u*v_2;
( 5,28)( 7,42)(18,23)(21,26)
gap> Image(hom2, w_2);
( 3,13)( 7,17)(19,20)(23,24)
```

w_2 の f_E での像は, 辺キューブ [7,24] と [17,23] を入れ換え, [3,19] と [13,20] をこの順序でいれ換える. そこで

```
gap> w_1^-1*w_2;
( 5,26)(21,28)
gap> Image(hom2, w_1^-1*w_2);
( 3,19)(13,20)
```

すなわち $w_1^{-1}w_2$ は最短距離にある 2 個の辺キューブを位置を動かさずにそれぞれ 180 度回転させる. 特に $w_1^{-1}w_2 \in (\mathbb{Z}/2\mathbb{Z})^{11}$ である. ルービック・キューブの対称性から最短距離にあるどんな 2 個の辺キューブについても同様の変換が (生成元を取り換えることにより) 構成できる. $(\mathbb{Z}/2\mathbb{Z})^{11}$ はこれらの変換により生成されるので, 結局辺キューブを (回転まで含めて) 全て正しい位置に持っていくことができる. これで $g \in H$ は生成元の積で表され, 同時にゲームが完成する.

6.6 拡張版ルービック・キューブ $(3 \times 3 \times 3)$

ルービック・キューブを構成している各キューブが色ではなく, 下図のように文字によって識別されているとする.

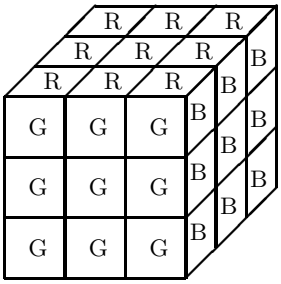


FIGURE 6. 拡張版ルービック・キューブ $(3 \times 3 \times 3)$

この場合, 頂点キューブと辺キューブについては前と変わりないが, 中心キューブについてはその位置での 90 度回転も考慮する必要が生じる. 今までの結果から, 通常のルービック・キューブ群 H は S_{48} の部分群として構成されている. そこで, 6 個の中心キューブの, それぞれ 90 度回転に合わせて, 24 個の文字, $\{49, 50, \ldots, 72\}$ を付け加え, S_{72} の部分群として拡張版ルービック・キューブ群 $\widetilde{H}$ を構成する. H の生成元 $x_1, \ldots, z_2$ に対応するルービック・キューブの操作によって各面の中心キューブの回転が誘導される. そこで $\widetilde{H}$ の生成元として `xx_1, xx_2, yy_1, yy_2, zz_1, zz_2` を次のようにおく.

```
gap> xx_1 := x_1*(49,50,51,52);;
gap> xx_2 := x_2*(53,54,55,56);;
gap> yy_1 := y_1*(57,58,59,60);;
gap> yy_2 := y_2*(61,62,63,64);;
gap> zz_1 := z_1*(65,66,67,68);;
gap> zz_2 := z_2*(69,70,71,72);;
```

そこで $\widetilde{H} =$ `excube` を以下のように定義する.

```
gap> excube := Group(xx_1, xx_2, yy_1, yy_2, zz_1, zz_2);
<permutation group with 6 generators>
gap> Size(excube);
88580102706155225088000
```

$H \times (\mathbb{Z}/4\mathbb{Z})^6$ の部分群 $\widetilde{H}$ の構造を決定しよう. $p : \widetilde{H} \to H$ を射影 $H \times (\mathbb{Z}/4\mathbb{Z})^6 \to H$ の $\widetilde{H}$ への制限とする. $p(xx_1) = x_1, \ldots, p(zz_2) = z_2$ より, p は全射準同型になる. $\widetilde{K} = \operatorname{Ker} p$ とおく. 次の完全系列を考える.

$$1 \longrightarrow \widetilde{K} \longrightarrow \widetilde{H} \longrightarrow H \longrightarrow 1 \tag{6.6.1}$$

ここで $\widetilde{K} = \widetilde{H} \cap (\mathbb{Z}/4\mathbb{Z})^6$ であり, また

```
gap> Size(excube)/Size(cube);
2048
gap> 4^6;
4096
```

より, $\widetilde{K}$ は $(\mathbb{Z}/4\mathbb{Z})^6$ の指数 2 の部分群になる. $\widetilde{K}$ の構造を調べる. $(\mathbb{Z}/4\mathbb{Z})^6$ は, 各面での 90 度回転に対応している. 今, 6 個の面の順番を隣り合った面は常に辺を共有するように定め, その生成元を $\varepsilon_1, \ldots, \varepsilon_6$ とする. 順番の決め方から $\varepsilon_1 = (49, 50, 51, 52)$ が x_1 に対応し, $\varepsilon_2 = (57, 58, 59, 60)$ が y_1 に対応するとしてよい. このとき

```
gap> (49,50,51,52) in excube;
false
gap> (49,50,51,52)(57,58,59,60) in excube;
true
gap> (49, 50,51,52)^2 in excube;
true
```

従ってルービック・キューブの対称性から, $\varepsilon_i^2 \in \widetilde{K}, \varepsilon_i\varepsilon_{i+1} \in \widetilde{K}$ が成り立つ. これより

$$\widetilde{K} = \left\{(\varepsilon_1^{\lambda_1}, \ldots, \varepsilon_6^{\lambda_6}) \in (\mathbb{Z}/4\mathbb{Z})^6 \mid \sum_{i=1}^{6} \lambda_i \equiv 0 \pmod 2\right\} \tag{6.6.2}$$

が得られる. これで一応 $\widetilde{H}$ の構造は分かったが, 完全系列 (6.6.1) は分裂しないので, (6.6.1) だけではやはり不満が残る. そこで $H \times (\mathbb{Z}/4\mathbb{Z})^6$ の部分群としての $\widetilde{H}$ のより精密な表示を追求してみる. まず次に注意する.

$$(\mathbb{Z}/3\mathbb{Z})^7 \times (\mathbb{Z}/2\mathbb{Z})^{11} \subset \widetilde{H}. \tag{6.6.3}$$

実際, $(11, 24, 8), (20, 15, 3) \in H_V$ は `[1..24]` $\Leftrightarrow$ `orbits[1]` により $(25, 19, 8), (11, 17, 6) \in S_{24}$ に対応し, $(7, 24), (14, 5) \in H_E$ は `[1..24]` $\Leftrightarrow$ `orbits[2]` により $(7, 18), (4, 10)$ に対応する. このとき

```
gap> (25,19,8)(11,17,6) in cube;
true
gap> (25,19,8)(11,17,6) in excube;
true
gap> (7,18)(4,10) in cube;
true
```

```
gap> (7,18)(4,10) in excube;
true
```

従って, $(25,19,8)(11,17,6)$, $(7,18)(4,10) \in \widetilde{H}$ が分かる. ルービック・キューブの対称性から, $(\mathbb{Z}/3\mathbb{Z})^7$, $(\mathbb{Z}/2\mathbb{Z})^{11}$ の生成元はすべて $\widetilde{H}$ に含まれる. よって (6.6.3) が成り立つ.

(6.6.3) より次が導かれる.

$$\widetilde{H} \simeq \widetilde{H}_1 \ltimes ((\mathbb{Z}/3\mathbb{Z})^7 \times (\mathbb{Z}/2\mathbb{Z})^{11})). \tag{6.6.4}$$

ここで $\widetilde{H}_1 = \widetilde{H} \cap (A(S_8 \times S_{12}) \times (\mathbb{Z}/4\mathbb{Z})^6)$ は $A(S_8 \times S_{12}) \times (\mathbb{Z}/4\mathbb{Z})^6$ の指数 2 の部分群である (半直積での $(\mathbb{Z}/4\mathbb{Z})^6$ の作用は自明な作用とする). $\widetilde{H}$ を決めるためには, $\widetilde{H}_1$ の構造を決めればよい. $A_8 \times A_{12}$ は $A(S_8 \times S_{12})$ の部分群であることに注意して次を示す.

$$A_8 \times A_{12} \subset \widetilde{H}_1. \tag{6.6.5}$$

まず $A(S_8 \times S_{12})$ の H への埋め込みを具体的に決めることから始める.

```
gap> Orbits(comp1, [1..24]);
[ [ 1, 2, 3, 4, 5, 13, 7, 8 ], [ 6, 14, 15, 16, 21, 17, 9, 11 ],
  [ 10, 18, 20, 12, 19, 23, 24, 22 ] ]
gap> Orbits(comp2, [1..24]);
[ [ 1, 2, 3, 4, 5, 6, 17, 13, 9, 12, 10, 7 ],
  [ 8, 16, 18, 19, 22, 14, 23, 20, 15, 21, 11, 24 ] ]
```

S_8 は `comp1` として H_V の中に実現された. 従って, S_8 は, 次の順序付けられた 8 個の組の置換として得られる.

$$[1,6,22],\ [2,14,18],\ [3,15,20],\ [4,16,12],\ [5,21,10],\ [13,17,19],\ [7,9,23],\ [8,11,24]$$

`orbits[1]` の上では, 対応表により

$$[1,9,35],\ [14,46,40],\ [17,11,6],\ [3,33,27],\ [48,32,38],\ [24,30,43],\ [22,41,16],\ [19,8,25]$$

となる. 同様に S_{12} は `comp2` として H_E の中に実現されるので, S_{12} は順序付けられた 12 個の組の置換として実現される.

$$[1,11],\ [2,16],\ [3,19],\ [4,22],\ [5,14],\ [6,8],\ [17,23],\ [13,20],\ [9,18],\ [12,15],\ [10,21],\ [7,24]$$

`orbits[2]` の上では

$$[2,34],\ [12,37],\ [5,26],\ [47,39],\ [10,4],\ [36,29],\ [42,23],\ [28,21],\ [44,15],\ [45,31],\ [13,20],\ [7,18]$$

となる. これより, 例えば 3 個の頂点キューブ [1,9,35], [14,46,40], [17,11,6] に関する長さ 3 の巡回置換は H の元として $(1,14,17)(9,46,11)(35,40,6)$ と表される. そこで

```
gap> (1,14,17)(9,46,11)(35,40,6) in cube;
true
gap> (1,14,17)(9,46,11)(35,40,6) in excube;
true
```

同様に 3 個の辺キューブ [2,34], [12,37], [5,26] に関する長さ 3 の巡回置換は H の元として $(2,12,5)(34,37,26)$ と表される. そこで

```
gap> (2,12,5)(34,37,26) in cube;
true
gap> (2,12,5)(34,37,26) in excube;
true
```

上の結果は S_8 と S_{12} の (少なくとも一つの) 長さ 3 の巡回置換が $\widetilde{H}$ に含まれることを意味する. ところで, $A(S_8\times S_{12})\times(\mathbb{Z}/4\mathbb{Z})^6$ は明らかに, $S_8\times S_{12}$ の共役の作用により不変である. $\widetilde{H}_1$ は $A(S_8\times S_{12})\times(\mathbb{Z}/4\mathbb{Z})^6$ の指数 2 の部分群なのでやはり, $S_8\times S_{12}$ の作用で不変になる. したがって S_8 のすべての長さ 3 の巡回置換, S_{12} のすべての長さ 3 の巡回置換が $\widetilde{H}_1$ に含まれることが分かる. 交代群は長さ 3 の巡回置換によって生成されるので, これより (6.6.5) が得られる.

さて (6.6.1) と (6.6.5) を合わせて, $\widetilde{H}_1$ は $(A_8\times A_{12})\times\widetilde{K}$ を指数 2 の部分群として含むことが分かる. 残りの剰余類の代表元は, 例えば

$$g=(1,14)(9,46)(35,40)\times(2,12)(34,37)\times(49,50,51,52)$$

で与えられる. 第 1 項が頂点キューブ [1,9,35] と [14,46,40] の互換, 第 2 項が辺キューブ [2,34] と [12,37] の互換, 第 3 項が中心キューブの 90 度回転である. これにより,

$$\widetilde{H}_1=\big(A_8\times A_{12})\times\widetilde{K}\big)\bigcup\big((A_8\times A_{12})\times\widetilde{K}\big)g \tag{6.6.6}$$

と表される. この結果か次のようにすれば, より見やすくなる. 今,

$$\begin{aligned}&\psi_1:A(S_8\times S_{12})\to A(S_8\times S_{12})/(A_8\times A_{12})\simeq\{\pm1\},\\&\psi_2:(\mathbb{Z}/4\mathbb{Z})^6\to(\mathbb{Z}/4\mathbb{Z})^6/\widetilde{K}\simeq\{\pm\}\end{aligned}$$

により 準同型 ψ_1,ψ_2 を定義する. 準同型 $\psi:A(S_8\times S_{12})\times(\mathbb{Z}/4\mathbb{Z})^6\to\{\pm1\}$ を $\psi(a,b)=\psi_1(a)\psi_2(b)$ により定義する. このとき, $\widetilde{H}_1=\mathrm{Ker}\,\psi$ が成り立つ. 以上の結果をまとめると,

(6.6.7) 拡張版ルービック・キューブ群 $\widetilde{H}$ は

$$\widetilde{H}\simeq\widetilde{H}_1\ltimes((\mathbb{Z}/3\mathbb{Z})^7\times(\mathbb{Z}/2\mathbb{Z})^{11})$$

で与えられる. ここに $\widetilde{H}_1 = \mathrm{Ker}\,\psi$ は $A(S_8 \times S_{12}) \times (\mathbb{Z}/4\mathbb{Z})^6$ の指数 2 の部分群である. 特に,

$$|\widetilde{H}| = \frac{1}{24}(8! \times 12! \times 3^8 \times 2^{12} \times 4^6) = 88580102706155225088000.$$

最後に, 拡張版ルービック・キューブ群 $\widetilde{H}$ の中心 $Z(\widetilde{H})$ を求めておく.

```
gap> Center(excube);
<permutation group of size 4096 with 10 generators>
```

従って $Z(\widetilde{H})$ の位数は 4096 である. g を前節のように H の中心の元とするとき

$$Z(\widetilde{H}) \simeq \langle g \rangle \times \widetilde{K} \tag{6.6.8}$$

が成り立つ. 実際

```
gap> g in Center(excube);
true
gap> (49,50,51,52)(57,58,59,60) in Center(excube);
true
gap> (49,50,51,52)^2 in Center(excube);
true
```

2 番目と 3 番目の式から, (6.6.2) を導いたのと同様の議論で $\widetilde{K} \subset Z(\widetilde{H})$ が云える. そこで $\langle g \rangle \times \widetilde{K}$ と $Z(\widetilde{H})$ の位数を比較して (6.6.8) を得る.