Mathematics 466 **Groups Generated by Elements** A. Hulpke

We have seen so far two ways of specifying subgroups: By listing explicitly all elements, or by specifying a defining property of the elements. In some cases neither variant is satisfactory to specify certain subgroups, and it is preferrable to specify a subgroup by generating elements.

Definition: Let G be a group and $a_1, a_2, \ldots, a_n \in G$. The set

$$\langle a_1, a_2, \ldots, a_n \rangle = \left\{ b_1^{\epsilon_1} \cdot b_2^{\epsilon_2} \cdot \cdots \cdot b_k^{\epsilon_k} \mid k \in N_0 = \{0, 1, 2, 3, , \ldots\}, b_i \in \{a_1, \ldots, a_n\}, \epsilon_i \in \{1, -1\} \right\}$$

(with the convention that for $k = 0$ the product over the empty list is e) is called the subgroup of G generated by $a_1, \ldots, a_n$.

Note: If G is finite, we can – similarly to the subgroup test – forget about inverses in the exponents.

Example: Let $G = S_4$ and $a_1 = (1,2)(3,4)$, $a_2 = (1,4)(2,3)$. Then

$$\langle a_1, a_2 \rangle = \{(), a_1, a_2, a_1 \cdot a_1 = (), a_1 \cdot a_2 = (1,3)(2,4), a_2 \cdot a_1^{-1} = (1,3)(2,4), a_1 \cdot a_2 \cdot a_1 = a_2, \ldots\}$$

(Careful: In this particular case $a_1 \cdot a_2 = a_2 \cdot a_1$, though S_4 is not abelian.) We find that regardless how long we form products, we never get elements other than $(), a_1, a_2, a_1 \cdot a_2$. Similarly, it is not hard to see that $S_4 = \langle (1,2), (1,2,3,4) \rangle$

Note: If you had already linear algebra you know this idea of "generation" in the form of spanning sets and bases. While the concept for groups is very similar, it is not possible to translate the concept of "dimension": Different (even "independent") generating sets for a subgroup in general will contain a different number of generators.

We now want to show that $\langle a_1, a_2, \ldots, a_n \rangle$ is always a subgroup:

Theorem: Let G be a group and $a_1, a_2, \ldots, a_n \in G$. Then $H := \langle a_1, a_2, \ldots, a_n \rangle \leq G$.
Proof: For $k = 0$ we get the identity, thus $e \in H$ and thus $H \neq \varnothing$. Now use the criterion for subgroups: Let $x, y \in H$. Then (by the definition of H) there exist $k, l \in \mathbb{N}_0$ and elements $b_i, c_j \in \{a_1, \ldots, a_n\}$, $\epsilon_i, \delta_j \in \{\pm 1\}$ such that:

$$x = b_1^{\epsilon_1} \cdot b_2^{\epsilon_2} \cdot \cdots \cdot b_k^{\epsilon_k}, \qquad y = c_1^{\delta_1} \cdot c_2^{\delta_2} \cdot \cdots \cdot c_l^{\delta_l}.$$

Note that $y^{-1} = c_l^{-\delta_l} \cdot c_{l-1}^{-\delta_{l-1}} \cdot \cdots \cdot c_1^{-\delta_1}$. Thus

$$x \cdot y^{-1} = b_1^{\epsilon_1} \cdot b_2^{\epsilon_2} \cdot \cdots \cdot b_k^{\epsilon_k} \cdot c_l^{-\delta_l} \cdot c_{l-1}^{-\delta_{l-1}} \cdot \cdots \cdot c_1^{-\delta_1},$$

which has the form of an element of H. Thus H is a subgroup.

Note: If $S \leq G$ is a subgroup with $a_1, \ldots, a_n \in S$, then certainly all those products must be elements of S (as S is closed under multiplication). Thus $\langle a_1, a_2, \ldots, a_n \rangle$ is the *smallest* subgroup of G containing all the elements $a_1, a_2, \ldots, a_n$.

Special Case: A special case is that of (sub)groups that can be generated by one generator. We call such a group *cyclic*. Then we have in the definition that $\langle a \rangle = \{b_1^{\epsilon_1} \cdot b_2^{\epsilon_2} \cdot \cdots \cdot b_k^{\epsilon_k}\}$ with $b_i \in \{a\}$. We can now use the rules for exponents and write everything as a single power of a. Thus we get:

$$\langle a \rangle = \{a^x \mid x \in \mathbb{Z}\} = \{a^0, a^1, a^{-1}, a^2, a^{-2}, a^3, a^{-3}, \ldots\}$$

is the set of all powers of a which agrees with the definition from the book.
Note that a cyclic group might be given by multiple (irredundant) generators. For example we have that

$$\langle (1,2), (3,4,5) \rangle = \langle (1,2)(3,4,5) \rangle$$

is cyclic.

How to calculate elements In some circumstances, it can be desirable to obtain an element list from generators. Since the operation in a group is associative, we have that

$$b_1^{\epsilon_1} \cdot b_2^{\epsilon_2} \cdot \cdots \cdot b_k^{\epsilon_k} = (b_1^{\epsilon_1} \cdot b_2^{\epsilon_2} \cdot \cdots \cdot b_{k-1}^{\epsilon_{k-1}}) \cdot b_k^{\epsilon_k}.$$

We can therefore obtain the elements by the following process in an recursive way:

> Starting with the identity, multiply all elements "so far" with all generators (and their inverses) until nothing new is obtained.

The following description of the algorithm is more formal:

1. Write down a "start" mark.
2. Write down the identity e (all products of length 0).
3. Mark the (current) end of the list with an "end" mark. Let x run through all elements in this list between the "start" and "end" mark.
4. Let y run through all the generators $a_1, \cdots a_n$.
5. Form the products $x \cdot y$ and $x \cdot y^{-1}$. If the result is not yet in the list, add it at the end.
6. If you have run through all x, y, and you find that you added new elements to the end of the list (after the "end" mark), make the "end" mark the "start" mark and go back to step 3.
7. Otherwise you have found all elements in the group.

If the group is finite at some point longer products must give the same result as shorter products (which have been computed before), thus the process will terminate.

Example: Let $G = U(48) = \{1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37, 41, 43, 47\}$ and $a_1 = 5$, $a_2 = 11$. We calculate that $a_1^{-1} = 29$, $a_2^{-1} = 35$.
We start by writing down the identity (S and E are start and end mark respectively):

$$S, 1, E$$

Now we form the products of all written down elements with a_1, a_2, a_1^{-1} and a_2^{-1}:

$$1 \cdot 5 = 5, 1 \cdot 11 = 11, 1 \cdot 29 = 29, 1 \cdot 35 = 35,$$

So our list now is $1, S, 5, 11, 29, 35, E$. Again (back to step 3) we run through all elements written down so far, and form products:

$$5 \cdot 5 = 25, 5 \cdot 11 = 7, \underline{5 \cdot 29 = 1}, 5 \cdot 35 = 31, \quad \underline{11 \cdot 5 = 7}, \underline{11 \cdot 11 = 25}, \underline{11 \cdot 29 = 31}, \underline{11 \cdot 35 = 1},$$
$$\underline{29 \cdot 5 = 1}, \underline{29 \cdot 11 = 31}, \underline{29 \cdot 29 = 25}, \underline{29 \cdot 35 = 7}, \quad \underline{35 \cdot 5 = 31}, \underline{35 \cdot 11 = 1}, \underline{35 \cdot 29 = 7}, \underline{35 \cdot 35 = 25}$$

The underlined entries are not new and thus did not get added again. Our new list therefore is $1, 5, 11, 29, 35, S, 25, 7, 31, E$. Again we form products:

$$\underline{25 \cdot 5 = 29}, \underline{25 \cdot 11 = 35}, \underline{25 \cdot 29 = 5}, \underline{25 \cdot 35 = 11}, \quad \underline{7 \cdot 5 = 35}, \underline{7 \cdot 11 = 29}, \underline{7 \cdot 29 = 11}, \underline{7 \cdot 35 = 5},$$
$$\underline{31 \cdot 5 = 11}, \underline{31 \cdot 11 = 5}, \underline{31 \cdot 29 = 35}, \underline{31 \cdot 35 = 29}$$

Now *no new* elements were found. Thus we got all elements of the subgroup, and we have $\langle 5, 11 \rangle = \{1, 5, 7, 11, 25, 29, 31, 35\}$. (The ordering of elements now is unimportant, as we list the subgroup as a set.)

Expression as words If we are giving a group by generators this means that each of its elements can be written – not necessarily uniquely – as a product of generators (and inverses). The process of enumerating all elements in fact gives such an expression. For example we trace back that $31 = 5 \cdot 11^{-1}$. For larger groups this of course can become very tedious, but it is a method that can easily be performed on a computer.

Example: (GAP calculation) Consider the permutation group generated by $(1, 2, 3, 4, 5)$ and $(1, 2)(3, 4)$:

```
gap> a1:=(1,2,3,4,5);
(1,2,3,4,5)
gap> a2:=(1,2)(3,4);
(1,2)(3,4)
gap> G:=Group(a1,a2);
Group([ (1,2,3,4,5), (1,2)(3,4) ])
gap> Elements(G);
```

```
[ (), (3,4,5), (3,5,4), (2,3)(4,5), (2,3,4), (2,3,5), (2,4,3), (2,4,5),
  (2,4)(3,5), (2,5,3), (2,5,4), (2,5)(3,4), (1,2)(4,5), (1,2)(3,4),
  (1,2)(3,5), (1,2,3), (1,2,3,4,5), (1,2,3,5,4), (1,2,4,5,3), (1,2,4),
  (1,2,4,3,5), (1,2,5,4,3), (1,2,5), (1,2,5,3,4), (1,3,2), (1,3,4,5,2),
  (1,3,5,4,2), (1,3)(4,5), (1,3,4), (1,3,5), (1,3)(2,4), (1,3,2,4,5),
  (1,3,5,2,4), (1,3)(2,5), (1,3,2,5,4), (1,3,4,2,5), (1,4,5,3,2), (1,4,2),
  (1,4,3,5,2), (1,4,3), (1,4,5), (1,4)(3,5), (1,4,5,2,3), (1,4)(2,3),
  (1,4,2,3,5), (1,4,2,5,3), (1,4,3,2,5), (1,4)(2,5), (1,5,4,3,2), (1,5,2),
  (1,5,3,4,2), (1,5,3), (1,5,4), (1,5)(3,4), (1,5,4,2,3), (1,5)(2,3),
  (1,5,2,3,4), (1,5,2,4,3), (1,5,3,2,4), (1,5)(2,4) ]
gap> Length(Elements(G));
60
gap> Order(G);
60
```

If we wanted to express an element of the group as a word in the generators, we could use the command `Factorization`. (This command in fact internally enumerates all elements and stores word expressions for each. the word returned therefore is as short as possible. The functionality however is limited by the available memory – if we cannot store all group elements it will fail.)
For example for the same group as before we express a (random) element and check that the result is the same:

```
gap> Factorization(G,(1,5)(3,4));
x1^-2*x2*x1^2
gap> a1^-2*a2*a1^2;
(1,5)(3,4)
```

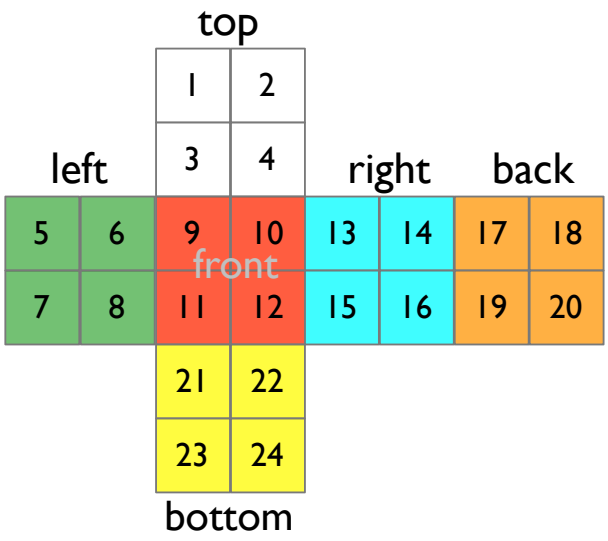
Application: Sort the sequence E, D, B, C, A by performing only swaps of adjacent elements: Considering the positions, we want to perform the permutation $(1,5)(2,4,3)$. The swaps of adjacent elements are the permutations $a_1 = (1,2)$, $a_2 = (2,3)$, $a_3 = (3,4)$, $a_4 = (4,5)$. Using GAP, we calculate:

```
gap> a1:=(1,2);
(1,2)
gap> a2:=(2,3);
(2,3)
gap> a3:=(3,4);
(3,4)
gap> a4:=(4,5);
(4,5)
gap> G:=Group(a1,a2,a3,a4);
Group([ (1,2), (2,3), (3,4), (4,5) ])
gap> Factorization(G,(1,5)(2,4,3));
x1*x2*x1*x3*x2*x4*x3*x2*x1
```

This means swap positions (in this order) 1/2, 2/3, 1/2, 3/4, 2/3, 4/5, 3/4, 2/3. 1/2 and you get the right arrangement (try it out!)

Puzzles Many puzzles can be described in this way: Each state of the puzzle corresponds to a permutation, the task of solving the puzzle then corresponds to expressing the permutation as a product of generators. Example 7 on p.107 in the book is an example of this.

The $2 \times 2 \times 2$ **Rubik's Cube in** GAP As an example of a more involved puzzle consider the $2 \times 2 \times 2$ Rubik's cube. We label the facelets of the cube in the following way:



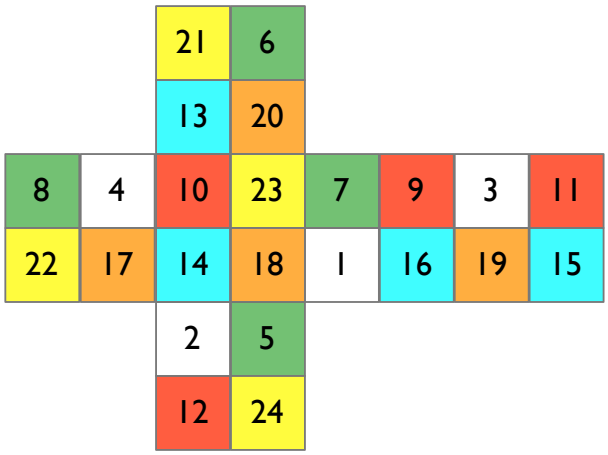
We now assume that we will fix the bottom right corner (i.e. the corner labelled with 16/19/24) in space – this is to make up for rotations of the whole cube in space. We therefore need to consider only three rotations, front, top and left. The coresponding permutations are (for clockwise rotation when looking at the face):

```
gap> top:=(1,2,4,3)(5,17,13,9)(6,18,14,10);;
gap> left:=(1,9,21,20)(5,6,8,7)(3,11,23,18);;
gap> front:=(3,13,22,8)(4,15,21,6)(9,10,12,11);;
gap> cube:=Group(top,left,front);
Group([(1,2,4,3)(5,17,13,9)(6,18,14,10),(1,9,21,20)(3,11,23,18)(5,6,8,7),
  (3,13,22,8)(4,15,21,6)(9,10,12,11) ])
gap> Order(cube);
3674160
```

By defining a suitable homomorphism first (for the time being consider this command as a black box – a free group is a group generated by formal symbols) we can choose nicer names – T, L and F – for the generators:

```
gap> map:=EpimorphismFromFreeGroup(cube:names:=["T","L","F"]);
[ T, L, F ] -> [ (1,2,4,3)(5,17,13,9)(6,18,14,10),
  (1,9,21,20)(3,11,23,18)(5,6,8,7), (3,13,22,8)(4,15,21,6)(9,10,12,11) ]
```

We now can use the command `Factorization` to express permutations in the group as word in generators. The *reverse* sequence of the inverse operations therefore will turn the cube back to its original shape. For example, suppose the cube has been mixed up in the following way:



This corresponds to the permutation

```
gap> move:=(1,15,20,4,6,2,21)(3,17,8,5,22,7,13)(9,14,11,18,12,23,10)
```

(1 has gone in the position where 15 was, 2 has gone in the position of 21, and so on.) We express this permutation as word in the generators:

```
gap> Factorization(cube,move);
T*F*L*T*F*T
```

We can thus bring the cube back to its original position by turning each *counter*clockwise top,front,top,left,front,top.

Larger puzzles If we want to do something similar for larger puzzles, for example the $3 \times 3 \times 3$ cube, the algorithm used by `Factorization` runs out of memory. Instead we would need to use a different algorithm, which can be selected by using the map we used for defining the name. The algorithm used then does not guarantee any longer a shortest word, in our example:

```
gap> PreImagesRepresentative(map,move);
T*L^-2*T^-1*L*T*L^-2*T^-2*F*T*F^-1*L^-1*F*T^-1*F^-1*L*T*L*
T^-2*F*T*F^-1*T*F*T^-1*F^-2*L^-1*F^2*L
```

(Note that the code uses some randomization, your mileage may vary.)

Web sites outside the mathematics department at Colorado State University that host this file without written permission by its Author, Alexander Hulpke, agree to pay the author $1 per day of hosting. This includes in particular any "PDF collection", "homework help", or "course notes" web sites.