Computational Group Theory

- the study of algorithms for groups
- devise algorithms to answer concrete questions about concrete groups.

given explicitly by generators "abstract"

given as symmetries of certain algebraic/geometric object "implicit".

Problems:

- Can we actually compute an abstract representation of implicit description? "symmetry detection"

  ↳ this is actually very hard!

- Can we actually perform calculations to obtain the objects defined in abstract algebra textbooks? (Can we compute the commutator subgroup? abelianization?)

- Can we compute something about this particular group? Eg. God's number for the Rubik Cube group?

- Algorithmic complexity ← if something is computable then how will the computations scale?

- Applications of group theoretical computations to other areas ("can we simplify this problem using it's symmetries?" e.g. chemistry ↔ crystallography, graph isomorphism)

What is the aim of this course?

- practical computability
- fast algorithms to be run on our computers
- Permutation groups
- Finitely presented groups

But NOT Matrix groups!
↳ highly efficient but also specialized algorithms for these exist.

Existing software:

- GAP (gap-system.org)
- Magma (magma.maths.usyd.edu.au)
- Sage (sagemath.org, also: cocalc.com) (run things interactively in a browser)

We will use neither :) instead we will start to develop our own software!

The aim: understand the problems by implementing the algorithms directly;

The counter-aim: Learn how to use some software that hides the algorithmic issues from us.

Exercises sessions:

- Learning how to code: language of my choice: julia (julialang.org)
- Learning how to
  - organize
  - test
  - maintain

  $\Bigg\}$ ⇒ reuse the code we've written.

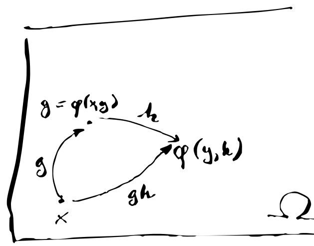
- Implementing concepts from lectures
- Band together in small teams, collaborate, but don't copy each other code!

1.1 Orbits and stabilizers

G – a group (non-empty set with binary operation usually denoted by $\cdot$, $e \leftarrow$ the identity element (sometimes also "1"), unary operation "$^{-1}$", axioms).

<u>Def</u>: G <u>acts</u> on a set Ω if there exists function $\varphi : \Omega \times G \longrightarrow \Omega$ satisfying:

- $\varphi(x,e) = x$ for all $x \in \Omega$
- $\varphi(\varphi(x,g),h) = \varphi(x,gh)$



this is so called <u>right action</u>.

Note most textbooks use **left** action:

$$\psi : G \times \Omega \longrightarrow \Omega \quad , \quad \psi(h, \psi(g,x)) = \psi(hg,x)$$

<u>Fact 1</u>

Each right action also defines left action via

$$\varphi(x,g) = \psi(g^{-1},x) \qquad \text{(and vice-versa).}$$

$$\varphi(\varphi(x,g),h) = \varphi(\psi(g^{-1},x),h) = \psi(h^{-1}, \psi(g^{-1},x)) =$$

$$= \psi(h^{-1}g^{-1},x) = \psi((gh)^{-1},x) = \varphi(x,gh).$$

Fact 2:

for each $g \in G$ $\quad \varphi_g : \Omega \to \Omega$

$$x \mapsto \varphi(x,g)$$

is a bijection.

(The inverse is just $\varphi_{g^{-1}}$)

Defn/Coroll. φ defines a group homomorphism

$$G \longrightarrow \mathrm{Sym}(\Omega)$$

$$g \longmapsto \varphi_g$$

known as the action homomorphism

Notation: we will be writing

x^g instead of $\varphi(x,g)$. Then the associativity law is just $(x^g)^h = x^{gh}$

Definition:

- Orbit of $x \in \Omega$ is $x^G = \{x^g : g \in G\} \subset \Omega$
- Stabilizer of $x \in \Omega$ is $\mathrm{Stab}_G(x) = \{g \in G : x^g = x\} \subset \Omega$.

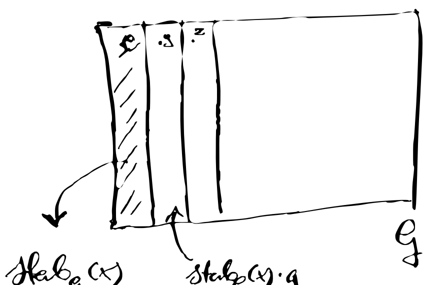
Exercise: $y^g \in x^G$ for every $y \in x^G$ and every $g \in G$.

Lemma:

- $Stab_G(x) < G$ (is a subgroup).
- there is a bijection of x^G and $\underbrace{Stab_G(x)\backslash G}_{\text{the set of right cosets}}$

Proof:

- $x^{gh} = (x^g)^h = x^h = x \Rightarrow gh \in Stab_G(x)$
- $x^{g^{-1}} = (x^g)^{g^{-1}} = x^{gg^{-1}} = x \Rightarrow g^{-1} \in Stab_G(x)$.



(partition of G into stabilizers

$$\mathcal{H} = Stab_G(x)$$

$$x^h = x \quad \forall h \in \mathcal{H}$$

$$x^{hg} = x^g \quad \forall h \in \mathcal{H}$$

every distinct point x^g on the orbit comes with $g^{-1}\mathcal{H}$ as its stabilizer, so

$$x^g \to g^{-1}\mathcal{H} = \mathcal{H}g$$

is a bijection

$$x^G \longleftrightarrow Stab_G(x)\backslash G$$

Corollary: $|x^G| = [G : Stab_G(x)]$ ← the index of $Stab_G(x)$ in G.

Computing an orbit:

Usually we don't have access to all elements of our group at once, we only know its generators.

ALGORITHM: (PLAIN ORBIT)

INPUT:
- S - a finite generating set for group G
- x - a point in Ω.

OUTPUT:
- x^G - the orbit of x under $\Omega \circlearrowleft G$

```
Δ := [x]
for δ in Δ
    for s ∈ S
        γ := δ^s
        if γ ∉ Δ
            Δ := Δ ∪ {γ}
        end
    end
end
return Δ
end
```

Note:
- Δ is modified (potentially) and "for δ in Δ" runs over the elements added to Δ as well
- This is correct when G is finite (otherwise we need to assume that the generating set is closed under taking inverses.

Performance note:

If $|S| = m$ and $|x^G| = n$ then

- δ^G will be computed $n \cdot m$ times
- checking that $y \in \Delta$ is a search problem:
    - * if Δ is sorted it will take $O(\log n)$ time
    - * if Δ is hashed (so called hash-table) then this might become so called "amortised $O(1)$"

Corollary:

The complexity of finding x^G is proportional to n, i.e. is $O(n)$.