Method Selection

GAP tries to select the "best" method (of several) for a particular operation (say `NormalSubgroups`) based on:

- What kind of object (Category — group of permutations)
- How is it stored (Representation — sparse polynomial)
- What is known (Attributes — solvable of order 1234)

It basically tries to select the most specialized method that still applies, assuming it is the best.

You can sometimes help GAP by explicitly setting information, in particular `Size` for large groups.

Creating Groups

There are three ways to create groups:

- Generators: Permutations, Matrices, automorphisms, ... The group is the *span* of these generators.
- Presentations: Words in generators, subject to relations (Finitely presented, special case: PcGroups) **Caveat:** Different groups are unrelated.
- Predefined Library of classes of groups GL, S_n, small order, transitive of small degree, simple,... Represented in appropriate ways.

Group Libraries, Constructors

Small Order (up to 2048, excluding 1024)

Transitive permutation groups, degree up to 30

Primitive permutation group, degree up to 2499

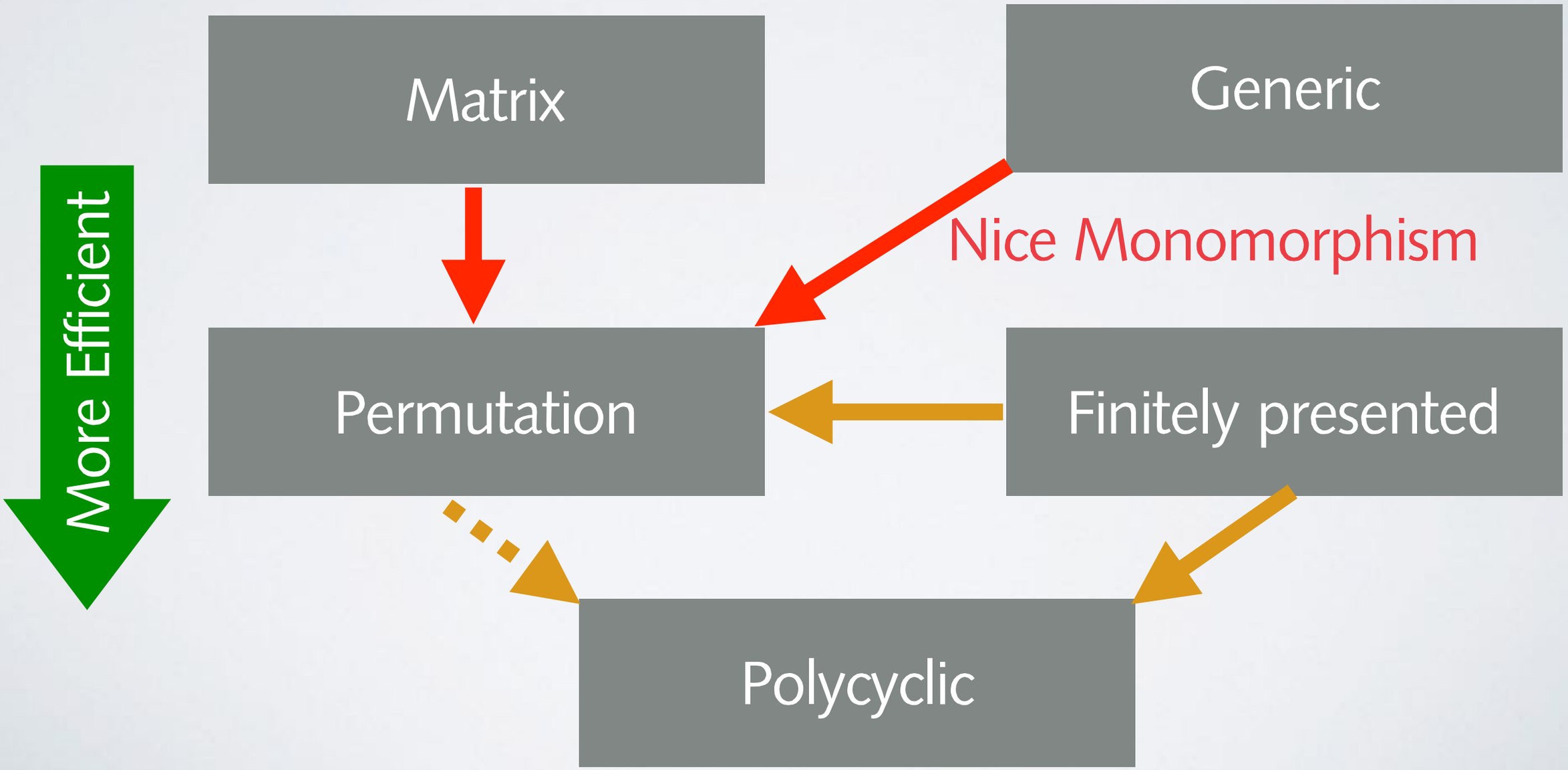
Perfect groups, Simple groups

Access as *Property*Group(*parameters*). First three also afford search All*Property*Groups, One*Property*Group with parameters property,value,property,value.

Constructors (such as AbelianGroup, DihedralGroup) often take category (e.g. IsPermGroup) as first argument.

Dependencies And Efficiency

Not all functionality is built in for every representation or works equally well — in some case GAP **will** (or **might**) calculate other representations and perform calculations there.



Converting Representations

Given a group in one representation it is possible to construct an isomorphic group in a different representation. (They return a map to the new representation).

IsomorphismPermGroup

IsomorphismFpGroup,
IsomorphismFpGroupByGenerators

IsomorphismPcGroup

IsomorphismSpecialPcGroup

Also NaturalHomomorphismByNormalSubgroup finds some way to represent factor group.

Homomorphisms

General Homomorphism operations:

- Image(hom,elt) or Image(hom,subgroup)
- ImagesRepresentative(hom,elt) (no membership test)
- PreImages(hom,group) or PreImagesRepresentative(hom,elt)
- KernelOfMultiplicativeGeneralMapping(hom)
- IsInjective, IsSurjective, etc.

Creating Homomorphisms

Besides the predefined homomorphisms to change representations:

- Homomorphisms to S_n induced by action on set.
- Specify generators and their images:

  `GroupHomomorphismByImages(from,to,gens,genimgs)` (will test and return `fail` if not a homomorphism. NC version)

Underlying Machinery

To work in a group we need to perform arithmetic with its elements and store them.

A permutation on n points takes roughly $n \cdot \log_2(n)$ bits.

$n=10^5$, 8GB memory: 40000 elements.

Thus store only *some* elements (represent a group by generators plus supporting data structures).

This is what we will look at now.

2

Orbits And Stabilizers

Basic Algorithmic Paradigms

- Linear Algebra
- Combinatorial Search
- Rewriting (String replacement)
- Group Actions

Group Actions

Assume that a group G acts on a set Ω, that is: g: $\omega \mapsto \omega^g$ is a bijection. Properties:

1. $\omega^1 = \omega$ where 1 is the identity
2. $\omega^{gh} = (\omega^g)^h$ (Here and in GAP action always from the right — matrices on row vectors)

e.g.Permutations on Points, Matrices on Vectors, Automorphisms

Orbit/Stabilizer theorem: Bijection between stabilizer cosets and orbit elements:

$$\mathrm{Stab}_G(\omega)\cdot g \leftrightarrow \omega^g$$

Computing With Actions

The natural algorithmic questions are to find:

ORBIT: ω^G of $\omega \in \Omega$. (Length, Elements).

STAB: Stabilizer of $\omega \in \Omega$ as subgroup (i.e. generators) of G.

TRANSPORTER: For $\omega, \delta \in \Omega$ find element $g \in G$ such that $\omega^g = \delta$ (or confirm that no such an element exists).

In all cases the group is given by generators.

Basic Orbit Algorithm

G acts on Ω. Orbit ω^G of $\omega \in \Omega$ consists of all images ω^g for $g \in G$.

Each g is a product of generators (and inverses if G is infinite – assume included)

Take iteratively images of all points obtained under all generators.

Cost: $|\omega^G| \cdot$#gens.

Input: $G = \langle g_1,...,g_m \rangle$, acting on Ω. Seed $\omega \in \Omega$.
Output: The orbit ω^G.

```
begin
  Δ := [ω];
  for δ ∈ Δ do
    for i ∈ {1,...,m} do
      γ := δ^(g_i);
      if γ ∉ Δ then
        Append γ to Δ;
      fi;
    od;
  od;
  return Δ;
end.
```

Modification: Transporters

Keeping track, we also get a *Transversal* of transporters $T[\delta]$, such that $\omega^{T[\delta]} = \delta$.

These $T[\delta]$ are also are reps. for (right) cosets of $Stab_G(\omega)$ in G.

If $\omega^g = \delta$ and $\omega^h = \gamma$, then $\delta^x = \gamma$ for $x = g^{-1}h$, solving the general transporter case.

```
begin
  Δ := [ω]; T := [1];
  for δ ∈ Δ do
    for i ∈ {1,...,m} do
      γ := δ^{g_i};
      if γ ∉ Δ then
        Append γ to Δ;
        Append T[δ] · g_i to T;
      fi;
    od;
  od;
  return Δ;
end.
```

Schreier's Lemma

If $\omega^a=\delta$, $\delta^b=\gamma$, $\omega^c=\gamma$, then $a\cdot b/c \in \mathrm{Stab}_G(\omega)$.

SCHREIER's lemma shows the converse, that is if we form such elements systematically, they generate $S=\mathrm{Stab}_G(\omega)$:

Lemma: $G=\langle \mathbf{g} \rangle$ finitely gen., $S\leq G$ with $[G:S]<\infty$.
Suppose $\mathbf{r}=\{r_1,\ldots,r_n\}$ a set of representatives for cosets of S in G, such that $r_1=1$.
For $h\in G$ write $\underline{h}:=r_i$ for the chosen coset representative, that is $Sr_i=Sh$ (i.e. $\omega^h=\omega^{r_i}$) Let

$$U:=\{r_i\, g_j(\,\underline{r_i\, g_j})^{-1} \mid r_i \in \mathbf{r},\, g_j \in \mathbf{g}\}$$

Then $S=\langle U \rangle$.
U is called the set of *Schreier generators* for S.

Modification: Stabilizer

The transversal gives coset representatives, the image of ω identifies cosets.

Thus we can form Schreier generators.

At this point

$S := \langle S,\ T[\delta] \cdot g_i\ /T[\gamma] \rangle$

just produces a generating set.

```
begin
  Δ := [ω]; T := [1]; S := ⟨1⟩;
  for δ ∈ Δ do
    for i ∈ {1,...,m} do
      γ := δ^{g_i};
      if γ ∉ Δ then
        Append γ to Δ;
        Append T[δ] · g_i to T;
      else
        S := ⟨S, T[δ] · g_i /T[γ]⟩;
      fi;
    od;
  od;
  return Δ;
end.
```

Consequences

The orbit algorithm and its variants let us solve

ORBIT, STABILIZER and TRANSPORTER as long as the orbit fits into memory.

By keeping track of the transversal, we write transversal elements as product of generators: HOMOMORPHISM.

If we let G act on itself this allows for element lists, centralizer, normalizer ***in small groups***.

To deal with larger cases, we need to use more group theory!

Group Actions in GAP

In GAP group actions are done by the operations:

`Orbit`, `Orbits`, `OrbitsDomain` (assumes closed)

`Stabilizer`, `RepresentativeAction` (Orbit/Stabilizer algorithm, sometimes backtrack, → lecture 2).

`Action` (Permutation image of action) and `ActionHomomorphism` (homomorphism to permutation image with image in symmetric group). (Also: `Permutation`)

The arguments are in general are:

A group G. (Will act by its `GeneratorsOfGroup`.)

A domain Ω (may be left out for `Orbit`, `Stabilizer`, but may improve performance).

Point ω, or list of point seeds for `Orbits`.

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> Orbit(g,1);
[ 1, 4, 2, 3 ]
gap> o:=Orbit(g,[1,2],OnSets);
[ [1, 2 ], [ 2, 3 ], [ 3, 4 ], [1,3], [1,4], [2,4] ]
gap> s:=Stabilizer(g,[3,4],OnSets);
Group([ (1,2), (3,4) ])
gap> RepresentativeAction(g,[3,4],[1,4],OnSets);
(1,2,3)
gap> RepresentativeAction(g,[1,3],[1,5],OnSets);
fail
gap> hom:=ActionHomomorphism(g,o,OnSets);
<action homomorphism>
gap> Orbits(Group((1,3),(1,2,3,4)),
> Combinations([1..4],2),OnSets);
[ [ [1,2],[2,3],[3,4], [1,4] ], [ [1,3], [2,4] ] ]
```

```
gap> a:=Action(g,o,OnSets);
Group([ (1,2,3,5)(4,6), (2,4)(5,6) ])
gap> hom:=ActionHomomorphism(g,o,OnSets);
<action homomorphism>
gap> a:=Action(g,Elements(g));
Group([ (2,22,7,3)(4,16,12,9)(5,21,20,13) […]
gap> Orbits(a,MovedPoints(a));
[[2,22,7,6,3,15],[4,16,12,9,20,13,5,21],[8,24,17], […]
  [ 10, 14, 18, 11, 23, 19 ] ]
gap> a:=Action(g,Elements(g),OnRight);
Group([ (1,10,17,19)(2,9,18,20)(3,12,14,21) […]
gap> RepresentativeAction(g,(1,2,3),(2,4,1));
(3,4)
gap> r:=List(ConjugacyClassesSubgroups(g),Representative);
[ Group(()), Group([ (1,3)(2,4) ]), Group([ (3,4) ]), […]
gap> all:=Concatenation(List(r,x->Orbit(g,x)));
[ Group(()), Group([(1,3)(2,4)]),Group([(1,4)(2,3)]), […]
gap> a:=Action(g,all);
Group([ (3,4)(5,6,7,9)(8,10)(11,12,13,14)(16,17) […]
```

Action functions

The last argument is an *action function* actfun(ω,g):=ω^g. This is a 2-argument GAP function that implements the actual action.
Some predefined actions are:

- OnPoints: action via ^. The default if not given.
- OnTuples, OnSets: Lists or sets (i.e. sorted lists) of points.
- OnSetsSets, OnSetsTuples, etc.
- OnRight: right multiplication *. (e.g. on cosets)
- OnLines: Projective on vectors: scaled first nonzero entry 1.
- OnSubspacesByCanonicalBasis: Subspaces, given as list of RREF basis vectors.
- Permuted: Permuting list entries.
- Your own function (a GAP function act(pnt,grpelm)), e.g. AsAutom:=function(sub,hom) return Image(hom,sub);end;

```
gap> g:=MathieuGroup(12);;S12:=SymmetricGroup(12);
gap> sets:=Combinations([1..12],6);;
gap> orb:=OrbitsDomain(g,sets,OnSets);;List(orb,Length);
[ 792, 132 ]
gap> Stabilizer(S12,orb[2],OnSetsSets);
Error, Action not well-defined. […]
gap> Stabilizer(S12,Set(orb[2]),OnSetsSets);
Group([ (1,11,10,9,8,7,6,5,4,3,2),(1,2,12, […]])
gap> Size(last);
95040
```

Optional Arguments

G may act via a homomorphism φ. (Say, a matrix group acting on enumerated vectors.) One can compute (in particular stabilizers) by giving two further list arguments:

gens A list of group generators,
imgs Images of these generators under φ.

Action Homomorphisms by default have codomain S_n. For large *n* this is inefficient. Append the string argument “surjective" to force the codomain equal to the image.

Action on cosets: Internal use of `PositionCanonical` (position of a *standard* equivalent object) allows:

`ActionHomomorphism(G,RightTransversal(G,U),OnRight, "surjective");` to get the action on the cosets of *U* in *G*.

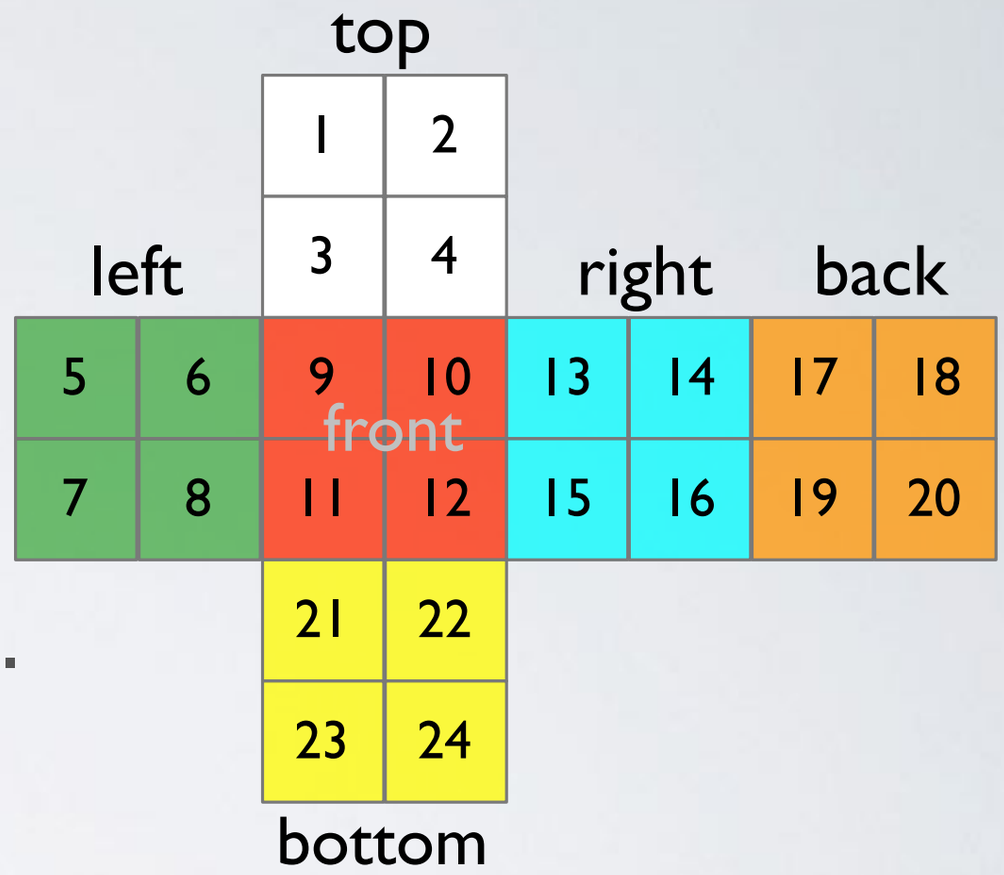
`FactorCosetAction` produces the same result.

Example: Rubik's Cube

For simplicity: 2×2×2.

Label faces with numbers, 1..24.

Fix bottom right corner (16/19/24).



Generators:

top:=(1,2,4,3)(5,17,13,9)(6,18,14,10);;

left:=(1,9,21,20)(5,6,8,7)(3,11,23,18);;

front:=(3,13,22,8)(4,15,21,6)(9,10,12,11);;

Orbit of 23

Start an Orbit calculation:

Point	23	18	14	3	10	1	11	13	6	12	2
Rep	e	L	LT	L^2	LT^2	L^2T	L^3	L^2F	LT^3	LT^2F	L^2T^2

Point	9	22	8	4	5	21	7	15	17	20
Rep	L^2TL	L^2F^2	L^2T^3L	L^2T^3F	L^2TLT	L^2TL^2	LT^3L^2	LT^3F^2	L^2TLT^2	L^2TL^3

E.g. $11^F=9$, so $L^3F/(L^2TL)$ lies in the stabilizer.

We find (subgroup order calculation) $\mathrm{Stab}_G(23)=\langle T,F,L^{-1}TL\rangle$,

Similarly $\mathrm{Stab}_G(22,23)=\langle T, L^{-1}TL, FTF^{-1}\rangle$ and

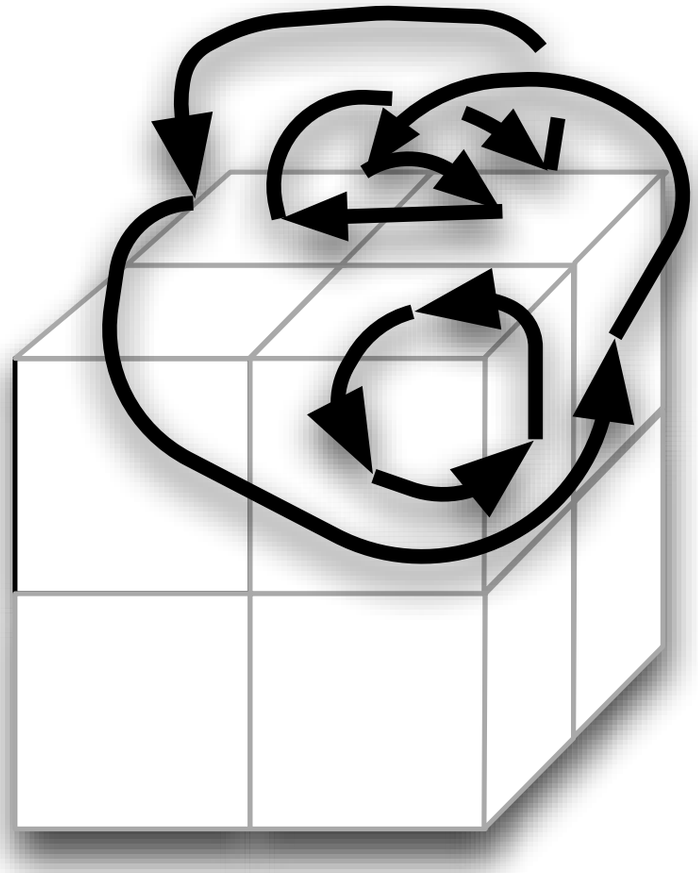
$\mathrm{Stab}_G(21,22,23)=\langle T, LT^{-1}L^{-1}T^{-1}F^{-1}LF\rangle$.

Orbit of 23

Start an Orbit calculation:

Point	23	18	14	3	10	1
Rep	e	L	LT	L^2	LT^2	L^2T

Point	9	22	8	4	5	2
Rep	L^2TL	L^2F^2	L^2T^3L	L^2T^3F	L^2TLT	L^2



(1,17,5,14,18,2)(4,10,13)

E.g. $11^F=9$, so $L^3F/(L^2TL)$ lies in the

We find (subgroup order calculatio

Similarly $\mathrm{Stab}_G(22,23)=\langle T, L^{-1}TL, FTF$

$\mathrm{Stab}_G(21,22,23)=\langle T, LT^{-1}L^{-1}T^{-1}F^{-1}LF \rangle$.

Brute-Force Solution

Use Orbit algorithm on *elements* to enumerate the group.
Transversal words then give (a) shortest representation.

```
cube:=Group(top,left,front);
map:=EpimorphismFromFreeGroup(cube:names:=["T","L","F"]);
Factorization(cube,(1,22,8)(2,17,14)(3,7,6,20,9,23)(5,12,11)
(15,21,18));
```

returns $L^{-1}T^{-1}FL^{2}F^{-1}L^{-1}F^{2}L^{-1}T$.

Limited by storage (here: ~10^{7} elements). Otherwise:

`PreImagesRepresentative(map,(1,22,8)(2,...)` returns

$T^{-1}F^{2}L^{-1}TL^{-1}T^{-1}FT^{-1}F^{-1}T^{-3}L^{-1}TLTL^{-1}TF^{-1}LFT^{-1}LT^{-1}$

Variants: Stabilizer Order

If storage or time requirements are an issue the following variants might help if $|G|$ is known:

- Known orbit length, partial stabilizer order can give early termination. If we can calculate subgroup orders, can stop if the largest proper divisor of $[G:S]$ is smaller than the orbit length.
- Use of Birthday paradox to estimate orbit length (coincidence at $\sqrt{|\Omega|}$) – indicate that full stabilizer is known.

More often than not I have ended up re-implementing an orbit algorithm instead of using a generic default...

Remarks on Performance

- Need to store whole orbit – memory limits scope.
- Store transversal T in factored form to save memory – *Schreier vector*. (Issue: balanced tree of low depth)
- Cost of basic algorithm is dominated by test $\gamma \in \Delta$? to check for new points – Data structures.
- There is a **huge** number of Schreier generators: Index of stabilizer $\times$ # group generators.

Usually many of them are redundant (or even trivial).

Dictionaries

To test $\gamma \in \Delta$? (and to determine $T[\gamma]$) one can

- Search linearly though Δ. (Orbit length n requires $\mathcal{O}(n^2)$ element comparisons)
- Keep a sorted copy of Δ. (needs $<$ test, $\mathcal{O}(n \log(n))$
- Determine index number for γ. (bit list, $\mathcal{O}(n)$)
- # Search in a hash table. (Hash key, almost $\mathcal{O}(n)$)

In GAP, the `Dictionary` data type provides a uniform interface.

All nontrivial approaches require dedicated handling for each data type. (Many objects do not have unique representations!)

Dictionaries

To test $\gamma \in \Delta$? (and to determine $T[\gamma]$) one can

- Search linearly though Δ. (Orbit length n requires $\mathcal{O}(n^2)$ element comparisons)
- Keep a sorted copy of Δ. (needs
- Determine index number for γ. (

Search in a hash table. (Hash key, almost $\mathcal{O}(n)$

Operation: `SparseIntKey(dom,elm)` to get hash key function.

In GAP, the `Dictionary` data type provides a uniform interface.

All nontrivial approaches require dedicated handling for each data type. (Many objects do not have unique representations!)

Reducing Schreier Generators

The number of Schreier generators cannot be reduced in general, as in free groups (later) all are needed.

- If membership test in partial stabilizer, test every new generator. (Does not produce minimal sets!)
- Randomization: Let $S=\{s_1,s_2,\ldots,s_n\}$ be a generating set. A *random subproduct* of S is a product $x=\prod_i s_i^{\epsilon_i}$ with the ϵ_i chosen independently by random from $\{0,1\}$.

Lemma: (BABAI, COOPERMAN, FINKELSTEIN, LUKS, SERESS, '95) Let $U=\langle S\rangle$. Subgroup chains of maximum length $m\leq\log_2(|U|)$. Then for every $\delta>0$ there exists a constant c , such that $c\cdot m$ random subproducts generate U with probability $1-\delta$.

3

Permutation Groups

Some Fundamental Tasks

*G*roups of permutations of degree up to a few 10^6, order easily 10^9 (so element orbit approach is infeasible), we want to solve:

ORDER: find the order of a group. (Implies element membership test.)

HOMOMORPHISM: decompose element as generator product. (Rewriting problem, Constructive Membership.)

We want to identify the groups composition STRUCTURE, possibly identify composition factors.

Also subgroup centralizers, normalizers, if index is huge.

```
gap> cube := Group(
> (1,3,8,6)(2,5,7,4)(9,33,25,17)(10,34,26,18)(11,35,27,19),
> (9,11,16,14)(10,13,15,12)(1,17,41,40)(4,20,44,37)…,…);;
<permutation group with 6 generators>
gap> Size(cube);
43252003274489856000
gap> DisplayCompositionSeries(cube);
G (6 gens, size 43252003274489856000)
  | Z(2)
S (12 gens, size 2162600163724492800O)
  | A(8) ~ A(3,2) = L(4,2) ~ D(3,2) = O+(6,2)
S (9 gens, size 1072718335180800)
  | Z(3)
    ⋮                7 copies in total
S (21 gens, size 490497638400)
  | A(12)
S (11 gens, size 2048)
  | Z(2)
    ⋮                  11 copies in total
1 (0 gens, size 1)
```

Use Subgroups

The principal idea now is to use subgroups/cosets to factor the problem: As $|G|=|U|\cdot[G:U]$ this logarithmizes the problem.

Suitable subgroups: Point stabilizers $U=\mathrm{Stab}_G(\omega)$, index at most $|\Omega|$.

We can iterate this process for U thus storage $|\Omega|^{\log_2(|G|)}$.

Caveat: This works for any group and any action (matrix, automorphism, etc.) but often the problem is that $[G:\mathrm{Stab}_G(\omega)]$ is not small.

Case in point: $\mathrm{GL}_n(q)$, orbit length q^n.

Use Subgroups

Lemma: Any subgroup chain $G>U_1>U_2>\ldots>U_k=\langle 1\rangle$ has at most $\log_2(|G|)$ steps.
Proof: If $U>V$, $U\neq V$, then $[U:V]\geqq 2$, so $|U|\geqq 2|V|$.
Thus $|G|\geqq 2^k$.

We can iterate this process for U thus storage $|\Omega|^{\log_2(|G|)}$.

Caveat: This works for any group and any action (matrix, automorphism, etc.) but often the problem is that $[G:\mathrm{Stab}_G(\omega)]$ is not small.

Case in point: $\mathrm{GL}_n(q)$, orbit length q^n.

Stabilizer Chains

Let $G \leq S_\Omega$. A list of points $B=(\beta_1,\ldots,\beta_m)$, $\beta_i \in \Omega$ is called a ***base***, if the identity is the only element $g \in G$ such that $\beta_i^g=\beta_i$ for all i.

The associated ***Stabilizer Chain*** is the sequence

$$G=G^{(0)} > G^{(1)} > \ldots > G^{(m)}= \langle 1 \rangle$$

defined by $G^{(0)}:=G$, $G^{(i)}:=\mathrm{Stab}_{G^{(i-1)}}(\beta_i)$. (Base property guarantees that $G^{(m)}= \langle 1 \rangle$.)

Note that every $g \in G$ is defined uniquely by base images $\beta_1^g,\ldots,\beta_m^g$. (If g,h have same images, then g/h fixes base.)

Base Length

The base length m often is short ($m \leq \log_2(|G|)$). In practice often $m < 10$.

We say that G is ***short-base*** if $\log|G| \leq \log^c|\Omega|$ for some c.

(Important to make polynomial in input size!)

Bounds on base length have been studied in theory. If there is no short base the groups must be essentially alternating A_n and relatives/products.

Same concept also possible for other kinds of groups, or different actions, but then no good orbit length/base length estimates.

Data structure

We will store for a stabilizer chain:

- The base points $(\beta_1,\ldots,\beta_m)$.
- Generators for all stabilizers $G^{(i)}$. (Union of all generators is ***strong generating set***, as it permits reconstruction of the $G^{(i)}$.) Data structure thus is often called **B**ase and **S**trong **G**enerating **S**et.
- The orbit of β_i under $G^{(i-1)}$ and an associated transversal for $G^{(i)}$ in $G^{(i-1)}$ (possibly as *Schreier tree*).

Storage cost thus is $\mathcal{O}(m \cdot |\Omega|)$

Data structure

We will store for a stabilizer chain:

- The base points $(\beta_1,\ldots,\beta_m)$.
- Generat[illegible] of all gener[illegible] permits recons[illegible] thus is often called [illegible] Strating Set.
- The orbit of β_i under $G^{(i-1)}$ and an associated transversal for $G^{(i)}$ in $G^{(i-1)}$ (possibly as *Schreier tree*).

Storage cost thus is $\mathcal{O}(m \cdot |\Omega|)$

Construction: Later

Consequences

- Group order: $G = [G^{(0)}:G^{(1)}]\cdot\ldots[G^{(m-1)}:G^{(m)}]$ and thus $G=\prod_i |\beta_i{}^{G(i-1)}|$.
- Membership test in G for $x \in S_\Omega$:

1. Is $\omega = \beta_1{}^x \in \beta_1{}^G$? If not, terminate.
2. If so, find transversal element $t \in G^{(0)}$ such that $\beta_1{}^t=\beta_1{}^x$.
3. Recursively: Is x/t (stabilizing β_1) in $G^{(1)}$.
   (Respectively: test x/y=( ) in last step.)

More Consequences

Bijection $g \in G \Leftrightarrow$ base image $(\beta_1{}^g, \beta_2{}^g, \ldots)$.

- Enumerate G, equal distribution random elements.
- Write $g \in G$ as product in transversal elts.
- Write $g \in G$ as product in strong generators.
- If strong gens. are words in group gens.: Write $g \in G$ in generators of G. (Caveat: Long words)
- Chosen base: Find stabilizers, transporter elements, for point tuples.

More Consequences

Bijection $g \in G \Leftrightarrow$ base image $(\beta_1{}^g, \beta_2{}^g, \ldots)$.

- Enumerate G, equal distribution random elements.
- Write g
- Write $g \in G$ as product in strong generators.

Computable Bijection $G \leftrightarrow \{1,2,\ldots,|G|\}$

- If strong gens. are words in group gens.: Write $g \in G$ in generators of G. (Caveat: Long words)
- Chosen base: Find stabilizers, transporter elements, for point tuples.

Generic Decomposition

remember:

```
cube:=Group(top,left,front);
map:=EpimorphismFromFreeGroup(cube:names:=["T","L","F"]);
Factorization(cube,(1,22,8)(2,17,14)(3,7,6,20,9,23)(5,12,11)
(15,21,18));
```

returned $L^{-1}T^{-1}FL^{2}F^{-1}L^{-1}F^{2}L^{-1}T$, but at high memory cost.

```
PreImagesRepresentative(map,(1,22,8)(2,17,14)(3,7,6,20,9,23)
(5,12,11)(15,21,18));
```

returns $T^{-1}F^{2}L^{-1}TL^{-1}T^{-1}FT^{-1}F^{-1}T^{-3}L^{-1}TLTL^{-1}TF^{-1}LFT^{-1}LT^{-1}$

Faster, less memory, but longer word.

Heuristics on trying short Schreier generators gets 3x3x3 cube word length to ~100 (naively: Millions)