GAP4
VI. PCGS

Alexander Hulpke

June 12, 1998

`http://www-gap.dcs.st-and.ac.uk/~ahulpke/course.html`

Abstract

The topic today is polycyclic groups and polycyclic generating systems. The latter permit to generalize solvable group algorithms to groups with solvable normal subgroups. The view taken here reflects my interests and is certainly biased: Pcgs are presented as a tool for building algorithms, the actual construction of Pcgs is taken as a "black box" operation.

Polycyclic groups

A group is *polycyclic* if it posesses a (finite) subnormal series

$$G = S_0 > S_1 > \cdots > S_n = \langle 1 \rangle$$

with S_i/S_{i+1} cyclic. The finite polycyclic groups are exactly the solvable groups. We can chose representatives g_i such that $S_{i-1}/S_i = \langle S_i s_i \rangle$. Then G is generated by the s_i. We can write every $g \in G$ uniquely in the form $g = \prod_{i=1}^{n} s_i^{a_i}$ with $0 \le a_i < e_i = |S_{i-1}/S_i|$.

We get a presentation of G in the s_i of the following form:

$$G = \langle s_1, \ldots, s_n \quad | \quad s_i^{e_i} = \prod_{j=i+1}^{n} s_j^{a_{ij}}, \quad [s_j, s_i] = \prod_{k=j}^{n} s_k^{b_{ijk}} \quad (j > i) \rangle$$

In fact for such presentations an algorithm exists to bring every word in its normal form ("Collection", PH. HALL).

Ususally one assumes the series to be refined such that e_i is a prime (or 0).

Vice versa every such presentation defines a group. (We however want that $|G| = \prod e_i$ such that the S_i form a proper series. This can be tested as well, "Consistency test".) In groups with such a presentation arithmetic is possible by applying collection.

In GAP groups of this type are called `PcGroups`. (The category `IsPcGroup` however is **not** a subcategory of `IsSubgroupFpGroup`.)

If G is a finite solvable group, `IsomorphismPcGroup` returns an isomorphism to a Pc group.

Constructing a Pc group from scratch is more complicated. To give a hint: `GapInputPcGroup(`*pcgroup*`,`*name*`)` returns a string that when read in by GAP produces an Pc group with the same presentation.

In a Pc group every element can be represented by an exponent vector $[e_1, \ldots, e_n]$, so storing elements is relatively cheap in comparison to the group size.

Vice versa every exponent vector determines an element of the Pc group.

PCGS

If we have a polycyclic group we can temporarily forget about the presentation and simply assign exponent vectors with respect to list of representatives s_i corresponding to an (arbitrary) subnormal series. We call this list $[s_1, \ldots, s_n]$ an *PCGS* (polycyclic generating system). In GAP a pcgs is implemented by an object that claims to be a list of the s_i but supports further operations.

A pcgs has the following properties:

1. $G = \langle s_1, \ldots, s_n \rangle$.
2. Every s_i has a *relative order* (the e_i): `RelativeOrders(`*pcgs*`)`.
3. Every $g \in G$ defines an exponent vector: `ExponentsOfPcElement(`*pcgs*`,`*elm*`)`, `LeadingExponentsOfPcElement(`*pcgs*`,`*elm*`)`. The index of the first nonzero entry is the `DepthOfPcElement(`*pcgs*`,`*elm*`)`.
4. Every exponent vector defines an element of G: `PcElementByExponents(`*pcgs*`,`*exp*`)`.
5. The subgroups $S_i := \langle s_j \mid j \ge i \rangle$ form a subnormal series: `PcSeries(`*pcgs*`)`.

The attribute `Pcgs` will returns a pcgs for a finite solvable group, for Pc groups `FamilyPcgs` returns the pcgs by which the group is defined.

Induced PCGS

If $U \le G$ and P is a pcgs for G, corresponding to a subnormal series S_i we can compute a subnormal series $U \cap S_i$ for U. We take coset representatives u_i for this series (eliminating trivial steps) such that the matrix of their exponent vectors is in echelon form.

Then the u_i form a pcgs for U, we say that it is induced by P. (If a generating set of U is given, a non-commutative GAUSS-elimination produces such an induced pcgs.)

In GAP, `InducedPcgs(`*pcgs*`,`*subgroup*`)` returns an induced pcgs *ipcgs*. From *ipcgs* we get *pcgs* back as `ParentPcgs(`*ipcgs*`)`.

(There are many more operations like `InducedPcgsByPcSeries`, as long as you do not start constructing pcgs yourself you will hardly need them.)

When algorithms use several pcgs they all must be induced by the same parent to be compatible.

For an induced pcgs *ipcgs* the attribute `CanonicalPcgs` returns another induced pcgs whose exponent matrix is in HERMITE normal form. It is uniquely determined by U.

Modulo PCGS

If $N \lhd G$ we can compute a pcgs Q for G/N. If N is a subgroup of the subnormal series for P, exponent vectors for Q are obtained from vectors for P by cutting off trailing entries of representatives.

We can do even better: If N is an arbitrary normal subgroup and I an induced pcgs with respect to P, we can take those elements of P that do not correspond to "steps" in I. They (the cosets represented by them) form a pcgs for G/N, the *modulo PCGS* $P \bmod I$. It consists of elements of G but computes exponent vectors with respect to the factor group.

Such a modulo pcgs has properties 2-4. (and 1 for the factor).

GAP computes a modulo pcgs by *pcgs* `mod` *ipcgs*. We can do something analoguous if N is not solvable by `ModuloPcgs(`*G*`,`*N*`)`.

Practically, modulo pcgs rid us of the need to construct factor groups.

For a modulo pcgs the attributes `NumeratorOfModuloPcgs` and `DenominatorOfModuloPcgs` return the dividend and divisor pcgs.

Special and *Special* PCGS

There are situations when we want a pcgs whose series is "nice". We might for example want that it refines a normal series with elementary abelian factors:

- `PcgsElementaryAbelianSeries` refines a series with elementary abelian factors,
- `PcgsCentralSeries` refines a central series,
- `PcgsPCentralSeriesPGroup` refines the p-central series of a p-group,
- `SpecialPcgs` runs through the LG-series [Eic94].

In all these cases `IndicesNormalSteps` gives the indices at which normal subgroups starts and `NormalSeriesByPcgs` returns the series of normal subgroups.

For Pc groups and permutation groups there are efficient algorithms to compute pcgs with these properties. They can differ from the `Pcgs` or `FamilyPcgs` of the group.

Algorithms using PCGS

A typical algorithm for solvable groups lifts the result via a series of elementary abelian normal subgroups. In each step the situation is $N \lhd G$ with N elementary abelian. The assumption is that the problem is solved in G/N, lifting then usually can be played down on the action of G on N (linear) or the combined affine action of G by conjugation and N by translation.

For example if $C/N = C_{G/N}(Ng)$, then C acts on Ng via

$$(ng)^c = n^c g^c = n^c c^{-1} g c = n^{c\underbrace{[c, g^{-1}]}_{\in N}} g$$

The stabilizer in this action is the centralizer.

The translation to pcgs is straightforward. We first compute a pcgs P corresponding to an elementary abelian series (probably with further properties). All elements of factor groups will be represented by representatives in the full group, subgroups will be represented by their full preimages.

When computing modulo $K \lhd G$, such that N/K is elementary abelian we take a `ModuloPcgs(`N`,`K`)`, exponents with respect to this pcgs reflect decomposition in this basis.

There are operations `LinearOperationLayer` and `AffineOperationLayer` that compute matrices for a given group G and a modulo pcgs.

The algorithm then only relies on the capability of the pcgs to yield exponent vectors. The *same* code will work for any type of group (Pc group, permutation group, ...), provided a pcgs is known.

Algorithms of this type are often more efficient than those that use the natural operation of a group (e.g. backtrack). Such algorithms exist for example for conjugacy classes, centralizers, normalizers, complements, subgroup lattice, invariant subgroups, maximal subgroups.

In fact often generalizations are possible if G is not solvable but posesses a solvable normal subgroup N. In this case a result for G/N can be lifted using the same approach.

Currently most algorithms require the series corresponding to a pcgs to be maximally refined, this can be tested by `IsPrimeOrderPcgs`.

Technical matters

Every finite solvable group has a pcgs, but the fact that we know that a group is solvable does not imply that we can compute a pcgs. For important classes of groups (permutation groups, Pc groups, groups that have a pcgs) this however holds.

To deal with this dilemma, GAP uses the filter `CanEasilyComputePcgs`. It is set by default for permutation groups or groups that have already a pcgs. It is inherited by subgroups. All algorithms that use a pcgs are installed with this filter as requirement.

The attribute `Pcgs` may return `fail` but this result should not imply `HasPcgs`: There is trickery with a method for 'SetPcgs'.

In fact `IsPcgs` is a subcategory of `IsModuloPcgs` which indicates that the subnormal series reaches the trivial subgroup. There is a further generalization, `IsGeneralizedPcgs` which only allows to compute relative orders but not necessarily depths or exponent vectors.

(Another aspect that will have to be represented better in future versions is that of *weights* of generators.)

The solvable orbit algorithm

If G acts transitively the orbits of a normal subgroup form a block system. This observation can help to improve orbit calculations:

Suppose we have computed the orbit of ω under $N \lhd G$ and want to compute the orbit ω^G. The N-orbit ω^N will be a block, so its images either are the same or intersect trivially (and then form another orbit of N).

This permits to execute the orbit algorithm with representatives that generate G/N acting on N.

If we have a subnormal series we can iterate this for every step. With a pcgs therefore every (sub-)orbit algorithm is done with only one acting element.

Similar tricks are possible for stabilizer and acting element.

In GAP an operation function `ExampleOperation`$(G, dom, gens, opgens, opfun)$ this algorithm is (usually) automatically used if *gens* is a pcgs.