FORMAT
A GAP Package on Formations

by

Bettina Eick and C.R.B. Wright

Fachbereich Mathematik und Informatik
der Technische Universität Braunschweig

and

University of Oregon

Bettina Eick
Fachbereich für Mathematik und Informatik, Technische Universität Braunschweig, 38106 Braunschweig, Germany
www-public.tu-bs.de:8080/∼beick
beick@tu-bs.de

Charles R.B. Wright
Department of Mathematics, University of Oregon, Eugene, Oregon 97403, U.S.A.
www.uoregon.edu/∼wright
wright@uoregon.edu

Contents

1 Introduction to FORMAT

The GAP package FORMAT provides functions to compute with formations of finite solvable groups. In addition to tools for constructing and combining formations, the package contains functions to compute $\mathscr{F}$-residual subgroups and to construct $\mathscr{F}$-normalizers and $\mathscr{F}$-covering subgroups determined by locally defined formations. System normalizers and Carter subgroups are available as special cases, and the $\mathscr{F}$-normalizer functions also apply to the computation of complements. The corresponding algorithms, together with applications and a complexity analysis, are described in [EW].

The package permits the computation of formation-theoretic subgroups not only for a number of classical formations, such as nilpotent, supersolvable or p-length 1 groups, but for other formations that the user may define. It also allows computation with classes of finite solvable groups defined by normal subgroup functions (see [DH], pages 395 ff). Attention may be restricted to the subgroups of a single group, a feature that has applications in the computation of complements to elementary abelian normal subgroups in finite solvable groups (see [EW]). An example of such an application is given in Section “format:other applications”.

This documentation contains only a brief account of the main formation-theoretic ideas. For a much more complete treatment we refer the reader to [DH]. Fundamental ideas of formation theory are described in [G] and [CH].

In the following sections we first describe the GAP definition of a formation and the examples of standard formations that are included in the package. We also present some functions that obtain new formations from ones already defined or that modify defined formations slightly. (See Section “format:formations in gap”.)

Then we describe functions that compute formation-theoretic subgroups of finite solvable groups (see Sections “format:residual functions”, “format:fnormalizers” and “format:covering subgroups”).

Finally we provide examples from a GAP session (see Sections “format:formation examples” and “format:other applications”) to illustrate the functions in the package.

2 Formations in GAP

A **formation** is a class $\mathscr{F}$ of groups closed under taking epimorphic images and subdirect products. Closure under subdirect products is equivalent to the property that each finite group G has a unique smallest normal subgroup $G^{\mathscr{F}}$ with factor group $G/G^{\mathscr{F}}$ in $\mathscr{F}$. The subgroup $G^{\mathscr{F}}$ is called the $\mathscr{F}$**-residual** subgroup of G. Thus, for example, the derived subgroup of G is its residual for the formation of abelian groups, and the residual for the formation of nilpotent groups is the last term of the descending central series.

In FORMAT a formation is described by a function that computes $G^{\mathscr{F}}$ for each (finite solvable) group G, and from that perspective $\mathscr{F}$ consists of the groups G for which $G^{\mathscr{F}}$ is trivial. To define a formation that is not one of the standard examples provided (see below), one must give GAP an identifier for the formation and also some method for computing residual subgroups.

Some of the most interesting formations can also be described by "local definition." For each prime p let $\mathscr{F}(p)$ be a formation or the empty class, and let $\mathscr{F}$ be the class of all finite solvable groups G such that for each prime p and each p-chief factor H/K of G the group of automorphisms that G induces on H/K by conjugation belongs to $\mathscr{F}(p)$. Then $\mathscr{F}$ is a formation, with **local definition** the set of $\mathscr{F}(p)$s. The set of primes p for which $\mathscr{F}(p)$ is not empty is called the **support** of $\mathscr{F}$. A p-chief factor is $\mathscr{F}$**-central** in case G induces an $\mathscr{F}(p)$-group on it or, equivalently, in case $G^{\mathscr{F}(p)}$ centralizes it. It is possible to define a formation in FORMAT by giving such a local definition. Indeed one can define a kind of generalized formation by giving what is called a normal subgroup function or **screen**, which specifies arbitrary normal subgroups, not necessarily of form $G^{\mathscr{F}(p)}$, to test "centrality." Section "format:other applications" describes one such usage of general screens. Most applications of formation theory to solvable groups require local definition, as do the GAP functions for computing $\mathscr{F}$-normalizers and $\mathscr{F}$-covering subgroups.

1 ► `Formation(` *rec* `)` O
► `Formation(` *str* `[,` *primes* `]` `)` O

The definition of a formation in FORMAT begins with the creation of a record `rec`, which must contain a `name` component and at least one of the components `fResidual` or `fScreen`. The component `name` is a string, `fResidual` is a function that computes a normal subgroup of each group, and `fScreen` is a function of two variables, a group and a prime, that returns a normal subgroup of the input group.

In the second form the function `Formation` can be used to obtain a formation from the supplied library of formations. The formations provided are:

`Formation( "Nilpotent" )`
The formation of nilpotent groups,

`Formation( "Supersolvable" )`
The formation of supersolvable groups,

`Formation( "Abelian" )`
The formation of abelian groups,

`Formation( "ElementaryAbelianProduct" )`
The formation of direct products of elementary abelian groups,

`Formation( "PNilpotent", prime )`
The formation of p-nilpotent groups for $p =$ `prime`,

`Formation( "PiGroups", primes )`
The formation of π-groups for $\pi =$ the set `primes`,

`Formation( "PLengthOne", prime )`
The formation of groups of p-length 1 for $p =$ `prime`.

2 ► `IsFormation( F )` C
► `NameOfFormation( F )` A
► `ResidualFunctionOfFormation( F )` A

`IsFormation` returns `true` if and only if F is a GAP formation. `NameOfFormation` returns the name of a formation and `ResidualFunctionOfFormation` returns the residual function of a formation.

3 ► `ScreenOfFormation( F )` A

If F is locally defined by some screen of $\mathscr{F}(p)$s, then `HasScreenOfFormation( F )` is `true`, `ScreenOfFormation( F )` is a function of two variables, *group* and *prime*, and `ScreenOfFormation( F )( G, p )` returns $G^{F(p)}$ if p is in the support of F and gives the empty list otherwise.

4 ► `SupportOfFormation( F )` A

The attribute `SupportOfFormation` is optional. It may be bound by `SetSupportOfFormation`. If `SupportOfFormation` is not bound, then the support of the formation is taken to be the set of all primes. In case the support of F is a finite set of primes, then `SupportOfFormation( F )` is a list of those primes, and `HasSupportOfFormation( F )` returns true. In case the support of F is an infinite set but not the set of all primes, then the user will need to make sure, perhaps with `ChangedSupport` or `SetSupportOfFormation`, that all primes dividing the orders of relevant groups are considered.

5 ► `ChangedSupport( F, primes )` O

This function may be used to change the support of a formation. Let F be a formation and *primes* a list of primes. Then `ChangedSupport` returns a formation with a new name whose support is the intersection of the support of F and *primes*.

6 ► `IsIntegrated( F )` P

The local definition is called **integrated** in case $\mathscr{F}(p)$ is contained in $\mathscr{F}$ for each prime p. The optional property `IsIntegrated` makes sense only if `HasScreenOfFormation( F )` is `true`. Notice that some of the functions described below will require that all of the attributes `HasScreenOfFormation( F )`, `HasIsIntegrated( F )` and `IsIntegrated( F )` are `true`. If unbound, this property can be bound with `SetIsIntegrated`, but it is up to the user to determine whether such a setting is appropriate. Section “format:formation examples” contains an example of such usage.

7 ► `Integrated( F )` O

A local definition of a formation may always be replaced by an integrated one without changing the formation itself, though the meaning of $\mathscr{F}$-central may change. Let F be a locally defined GAP formation with name *name*. If F is already integrated, then `Integrated( F )` yields F itself. Otherwise, it yields a formation *name*`Int` that is abstractly the same as F but has integrated local definition.

8 ► *F1* `=` *F2*
► *F1* `<` *F2*

Two GAP formations *F1* and *F2* are considered to be equal in case they have the same name. The natural ordering on strings gives an ordering on formations. This ordering is useful for organizing key-dependent lists but has no mathematical significance.

9 ► `Intersection( F1, F2 )` O

The intersection of two GAP formations *F1* and *F2* is again a formation. `Intersection` produces the new formation (*name1*`And`*name2*), which has attribute `ResidualFunctionOfFormation` if either *F1* or *F2* does, has `FScreen` whenever both formations have `FScreen`, and is integrated if both are.

10 ► `ProductOfFormations( F1, F2 )` O

The product of two formations *F1* and *F2* is the formation *F* such that a finite group *G* is a member of *F* if and only if G^{F2} is in *F1*. (Notice that the product of *F1* by *F2* is not necessarily equal to the product of *F2* by *F1*, and unless *F1* is normal subgroup-closed the product need not contain all extensions of a group in *F1* by a group in *F2*.) The function `ProductOfFormations( F1, F2 )` yields the product (*name1*`By`*name2*) of the two formations. The product has the attribute `ResidualFunctionOfFormation` and has the attribute `ScreenOfFormation` whenever both *F1* and *F2* have this entry or whenever both `HasScreenOfFormation( F2 )` and `not HasSupportOfFormation( F1 )` are `true`. In these cases the property `IsIntegrated` will be inherited if possible.

3 Residual Functions

1 ► `ResidualWrtFormation( G, F )` O

Let G be a finite solvable group and F a formation. Then `ResidualWrtFormation` returns the F-residual subgroup of G.

The following special cases have their own functions.

2 ► `NilpotentResidual( G )` A

This is the last term of the descending central series of G.

3 ► `PResidual( G, p )` O

This is the smallest normal subgroup of G whose index is a power of the prime p.

4 ► `PiResidual( G, primes )` O

This is the smallest normal subgroup of G whose index is divisible only by primes in the list *primes*.

5 ► `CoprimeResidual( G, primes )` O

This is the smallest normal subgroup of G whose index is divisible only by primes **not** in the list `primes`.

6 ► `ElementaryAbelianProductResidual( G )` A

This is the smallest normal subgroup of G whose factor group is a direct product of groups of prime order.

4 FNormalizers

Let $\mathscr{F}$ be an integrated locally defined formation, and let G be a finite solvable group with Sylow complement basis $\Sigma := \{S^p \mid p \text{ divides } |G|\}$. Let π be the set of prime divisors of the order of G that are in the support of $\mathscr{F}$ and ν the remaining prime divisors of the order of G. Then the **$\mathscr{F}$-normalizer** of G with respect to Σ is defined to be $\bigcap_{q\in\nu} S^q \cap \bigcap_{p\in\pi} N_G(G^{\mathscr{F}(p)} \cap S^p)$. The special case $\mathscr{F}(p) = \{1\}$ for all p defines the formation of nilpotent groups, whose $\mathscr{F}$-normalizers are the **system normalizers** of G. The $\mathscr{F}$-normalizers of a group G for a given $\mathscr{F}$ are all conjugate. They cover $\mathscr{F}$-central chief factors and avoid $\mathscr{F}$-hypereccentric ones.

1 ► `FNormalizerWrtFormation( G, F )` O
► `SystemNormalizer( G )` A

If F is a locally defined integrated formation in GAP and G is a finite solvable group, then the function `FNormalizerWrtFormation` returns an F-normalizer of G. The function `SystemNormalizer` yields a system normalizer of G.

The underlying algorithm here requires G to have a special pcgs (see section 46 in the GAP reference manual), so the algorithm's first step is to compute such a pcgs for G if one is not known. The complement basis Σ associated with this pcgs is then used to compute the F-normalizer of G with respect to Σ. This process means that in the case of a finite solvable group G that does not have a special pcgs, the first call of `FNormalizerWrtFormation` (or similarly of `FormationCoveringGroup`) will take longer than subsequent calls, since it will include the computation of a special pcgs.

The `FNormalizerWrtFormation` algorithm next computes an F-system for G, a complicated record that includes a pcgs corresponding to a normal series of G whose factors are either F-central or F-hypereccentric. A subset of this pcgs then exhibits the F-normalizer of G determined by Σ. The list `ComputedFNormalizerWrtFormations( G )` stores the F-normalizers of G that have been found for various formations F.

The `FNormalizerWrtFormation` function can be used to study the subgroups of a single group G, as illustrated in an example in Section "format:other applications". In that case it is sufficient to have a function `ScreenOfFormation` that returns a normal subgroup of G on each call.

5 Covering Subgroups

Let $\mathscr{X}$ be a collection of groups closed under taking homomorphic images. An **$\mathscr{X}$-covering subgroup** of a group G is a subgroup E satisfying

(C) $E \in \mathscr{X}$, and $EV = U$ whenever $E \leq U \leq G$ with $U/V \in \mathscr{X}$.

It follows from the definition that an $\mathscr{X}$-covering subgroup E of G is also $\mathscr{X}$-covering in every subgroup U of G that contains E, and an easy argument shows that E is an **$\mathscr{X}$-projector** of every such U, i.e., E satisfies

(P) EK/K is an $\mathscr{X}$-maximal subgroup of U/K whenever K is normal in U.

Gaschütz showed that if $\mathscr{F}$ is a locally defined formation, then every finite solvable group has an $\mathscr{F}$-covering subgroup. Indeed, locally defined formations are the only formations with this property. For such formations the $\mathscr{F}$-projectors and $\mathscr{F}$-covering subgroups of a solvable group coincide and form a single conjugacy class of subgroups. (See [DH] for details.)

1 ► `CoveringSubgroup1( G, F )` O
► `CoveringSubgroup2( G, F )` O
► `CoveringSubgroupWrtFormation( G, F )` O

If F is a locally defined integrated formation in GAP and if G is a finite solvable group, then the command `CoveringSubgroup1( G, F )` returns an F-covering subgroup of G. The function `CoveringSubgroup2` uses a different algorithm to compute $\mathscr{F}$-covering subgroups. The user may choose either function. Experiments with large groups suggest that `CoveringSubgroup1` is somewhat faster. `CoveringSubgroupWrtFormation` checks first to see if either of these two functions has already computed an F-covering subgroup of G and, if not, it calls `FCoveringGroup1` to compute one.

Nilpotent-covering subgroups are also called **Carter subgroups**.

2 ► `CarterSubgroup( G )` A

The command `CarterSubgroup( G )` is equivalent to `CoveringSubgroupWrtFormation( G, Formation( "Nilpotent" ) )`.

All of these functions call upon $\mathscr{F}$-normalizer algorithms as subroutines.

6 Formation Examples

The following is a GAP session that illustrates the various functions in the package. We have chosen to work with the symmetric group S_4 and the special linear group $SL(2,3)$ as examples, because it is easy to print and read the results of computations for these groups, and the answers can be checked by inspection. However, both S_4 and $SL(2,3)$ are extremely small examples for the algorithms in FORMAT. In [EW] we describe effective application of the algorithms to groups of composition length as much as 61, for which the computations take a few seconds to complete. The file `grp` contains some of these groups and other groups readable as GAP input.

```
gap> LoadPackage("format");;
```

A primitive banner appears.

First we define S_4 as a permutation group and compute some subgroups of it.

```
gap> G := SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> SystemNormalizer(G);  CarterSubgroup(G);
Group([ (3,4) ])
Group([ (3,4), (1,3)(2,4), (1,2)(3,4) ])
```

Now we take the formation of supersolvable groups from the examples and look at it.

```
gap> sup := Formation("Supersolvable");
formation of Supersolvable groups
gap> KnownAttributesOfObject(sup); KnownPropertiesOfObject(sup);
[ "NameOfFormation", "ScreenOfFormation" ]
[ "IsIntegrated" ]
```

We can look at the screen for `sup`.

```
gap> ScreenOfFormation(sup);
<Operation "AbelianExponentResidual">
gap> ScreenOfFormation(sup)(G,2); ScreenOfFormation(sup)(G,3);
Group([ (3,4), (2,4,3), (1,4)(2,3), (1,3)(2,4) ])
Group([ (2,4,3), (1,4)(2,3), (1,3)(2,4) ])
```

We get the residuals for `G` of the formations of abelian groups of exponent 1 ($= 2 - 1$) and of exponent 2 (=$3 - 1$).

Notice that `sup` does not yet have a residual function. Let's compute some subgroups of `G` corresponding to `sup`.

```
gap> ResidualWrtFormation(G, sup);
Group([ (1,2)(3,4), (1,4)(2,3) ])
gap> KnownAttributesOfObject(sup);
[ "NameOfFormation", "ScreenOfFormation", "ResidualFunctionOfFormation" ]
```

The residual function for `sup` was required and created.

```
gap> FNormalizerWrtFormation(G, sup);
Group([ (3,4), (2,4,3) ])
gap> CoveringSubgroupWrtFormation(G, sup);
Group([ (3,4), (2,4,3) ])
gap> KnownAttributesOfObject(G);
[ "Size", "OneImmutable", "SmallestMovedPoint", "NrMovedPoints",
  "MovedPoints", "GeneratorsOfMagmaWithInverses", "TrivialSubmagmaWithOne",
  "MultiplicativeNeutralElement", "DerivedSubgroup", "IsomorphismPcGroup",
  "IsomorphismSpecialPcGroup", "PcgsElementaryAbelianSeries", "Pcgs",
  "GeneralizedPcgs", "StabChainOptions", "ComputedResidualWrtFormations",
  "ComputedAbelianExponentResiduals", "ComputedFNormalizerWrtFormations",
  "ComputedCoveringSubgroup1s", "ComputedCoveringSubgroup2s",
  "SystemNormalizer", "CarterSubgroup" ]
```

The `AbelianExponentResiduals` were computed in connection with the local definition of `sup`. (`AbelianExponentResidual(G, n)` returns the smallest normal subgroup of `G` whose factor group is abelian of exponent dividing `n-1`.) Here are some of the other records.

```
gap> ComputedResidualWrtFormations(G);
[ formation of Supersolvable groups , Group([ (1,2)(3,4), (1,4)(2,3) ]) ]
gap> ComputedFNormalizerWrtFormations(G);
[ formation of Nilpotent groups , Group([ (3,4) ]),
  formation of Supersolvable groups , Group([ (3,4), (2,4,3) ]) ]
gap> ComputedCoveringSubgroup2s(G);
[  ]
gap> ComputedCoveringSubgroup1s(G);
[ formation of Nilpotent groups , Group([ (3,4), (1,3)(2,4), (1,2)(3,4) ]),
  formation of Supersolvable groups , Group([ (3,4), (2,4,3) ]) ]
```

The call by `CoveringSubgroupWrtFormation` was to `CoveringSubgroup1`, not `CoveringSubgroup2`.

We could also have started with a pc group or a nice enough matrix group.

```
gap> s4 := SmallGroup(IdGroup(G));
<pc group of size 24 with 4 generators>
```

This is S_4 again. The answers just look different now.

```
gap> SystemNormalizer(s4); CarterSubgroup(s4);
Group([ f1 ])
Group([ f1, f4, f3*f4 ])
```

Similarly, we have $SL(2,3)$ and an isomorphic pc group.

```
gap> sl := SpecialLinearGroup(2,3);
SL(2,3)
gap> h := SmallGroup(IdGroup(sl));
<pc group of size 24 with 4 generators>
```

We get the following subgroups.

```
gap> CarterSubgroup(sl); Size(last);
<group of 2x2 matrices in characteristic 3>
6
gap> SystemNormalizer(h); CarterSubgroup(h);
Group([ f1, f4 ])
Group([ f1, f4 ])
```

Now let's make new formations from old.

```
gap> ab := Formation("Abelian");
formation of Abelian groups
gap> KnownPropertiesOfObject(ab); KnownAttributesOfObject(ab);
[  ]
[ "NameOfFormation", "ResidualFunctionOfFormation" ]
gap> nil2 := Formation("PNilpotent",2);
formation of 2Nilpotent groups
gap> KnownPropertiesOfObject(nil2); KnownAttributesOfObject(nil2);
[ "IsIntegrated" ]
[ "NameOfFormation", "ScreenOfFormation", "ResidualFunctionOfFormation" ]
```

Compute the product and check some attributes.

```
gap> form := ProductOfFormations(ab, nil2);
formation of (AbelianBy2Nilpotent) groups
gap> KnownAttributesOfObject(form);
[ "NameOfFormation", "ResidualFunctionOfFormation" ]
```

Now the product in the other order, which **is** locally defined.

```
gap> form2 := ProductOfFormations(nil2, ab);
formation of (2NilpotentByAbelian) groups
gap> KnownAttributesOfObject(form2);
[ "NameOfFormation", "ScreenOfFormation", "ResidualFunctionOfFormation" ]
```

We check the results on `G`, which is still S_4.

```
gap> ResidualWrtFormation(G, form);  ResidualWrtFormation(G, form2);
Group(())
Group([ (1,3)(2,4), (1,2)(3,4) ])
gap> KnownPropertiesOfObject(form2);
[  ]
```

Although `form2` is not integrated, we can make an integrated formation that differs from `form2` only in its local definition, i.e., whose residual subgroups are the same as those for `form2`.

```
gap> Integrated(form2);
formation of (2NilpotentByAbelian)Int groups
```

`FNormalizerWrtFormation` and `CoveringSubgroupWrtFormation` both require integrated formations, so they silently replace `form2` by this last formation without, however, changing `form2`.

```
gap> FNormalizerWrtFormation(G, form2); CoveringSubgroupWrtFormation(G, form2);
Group([ (3,4), (2,4,3) ])
Group([ (3,4), (2,4,3) ])
gap> KnownPropertiesOfObject(form2);
[  ]
gap> ComputedCoveringSubgroup1s(G);
[ formation of (2NilpotentByAbelian)Int groups , Group([ (3,4), (2,4,3) ]),
  formation of Nilpotent groups , Group([ (3,4), (1,3)(2,4), (1,2)(3,4) ]),
  formation of Supersolvable groups , Group([ (3,4), (2,4,3) ]) ]
gap> ComputedResidualWrtFormations(G);
[ formation of (2NilpotentByAbelian) groups ,
  Group([ (1,4)(2,3), (1,2)(3,4) ]),
  formation of (AbelianBy2Nilpotent) groups , Group(()),
  formation of 2Nilpotent groups , Group([ (1,2)(3,4), (1,3)(2,4) ]),
  formation of Abelian groups , Group([ (2,4,3), (1,4)(2,3), (1,3)(2,4) ]),
```

```
    formation of Supersolvable groups , Group([ (1,2)(3,4), (1,4)(2,3) ]) ]
```

Lots of work has been going on behind the scenes.

Before we compute an intersection, we construct yet another formation.

```
gap> pig := Formation("PiGroups", [2,5]);
formation of (2,5)-Group groups with support [ 2, 5 ]
gap> form := Intersection(pig, nil2);
formation of ((2,5)-GroupAnd2Nilpotent) groups with support [ 2, 5 ]
gap> KnownAttributesOfObject(form);
[ "NameOfFormation", "ScreenOfFormation", "SupportOfFormation",
  "ResidualFunctionOfFormation" ]
```

Let's cut down the support of `nil2` to $\{2, 5\}$.

```
gap> form3 := ChangedSupport(nil2, [2,5]);
formation of Changed2Nilpotent[ 2, 5 ] groups
gap> SupportOfFormation(form3);
[ 2, 5 ]
gap> form = form3;
false
```

Although the formations defined by `form` and `form3` are abstractly identical, GAP has no way to know this fact, and so distinguishes them.

We can mix the various operations, too.

```
gap> ProductOfFormations(Intersection(pig, nil2), sup);
formation of (((2,5)-GroupAnd2Nilpotent)BySupersolvable) groups
gap> Intersection(pig, ProductOfFormations(nil2, sup));
formation of ((2,5)-GroupAnd(2NilpotentBySupersolvable)) groups with support
[ 2, 5 ]
```

Now let's define our own formation.

```
gap> preform := rec( name := "MyOwn",
>  fScreen := function( G, p)
>  return DerivedSubgroup( G );
>  end);
rec( fScreen := function( G, p ) ... end, name := "MyOwn" )
gap> form := Formation(preform);
formation of MyOwn groups
gap> KnownAttributesOfObject(form); KnownPropertiesOfObject(form);
[ "NameOfFormation", "ScreenOfFormation" ]
[  ]
```

In fact, the definition is integrated. Let's tell GAP so and compute some related subgroups.

```
gap> SetIsIntegrated(form, true);
gap> ResidualWrtFormation(G, form);
Group([ (1,4)(2,3), (1,2)(3,4) ])
gap> FNormalizerWrtFormation(G, form);
Group([ (3,4), (2,4,3) ])
gap> CoveringSubgroup1(G, form);
Group([ (3,4), (2,4,3) ])
```

These answers are consistent with the fact that `MyOwn` is really just the formation of abelian by nilpotent groups.

7 Other Applications

Up to this point our screens, i.e., normal subgroup functions, have yielded local formation residual subgroups, but there is no requirement that they do so. Screens for which the selected normal subgroups can be arbitrary have applications beyond formation theory. Chapter V of [CH] contains an account of a generalized normalizer theory built from them, and Wright ([WA] and [WB]) uses them to construct internal versions of formations that are conceptually related to ordinary formations much as Fitting sets are related to Fitting classes.

A major application of the generalized normalizers is to speed up computation of complements to normal factors (see [EW]). Suppose that G is a finite solvable group with an elementary abelian normal subgroup A for which there exists a normal subgroup N of G containing A such that N/A is nilpotent and $[N,A] = A$. Then A has a complement in G, and all complements are conjugate—indeed, they can be viewed as generalized $\mathscr{F}$-normalizers. We will show the idea, which of course is most useful with very large groups, by using `FNormalizerWrtFormation` to find a complement to an elementary abelian normal subgroup, in this case to K in S_4 with $N = A_4$.

We need to define a formation `F` in GAP (not a real formation, of course, just a local version) such that `ScreenOfFormation( F )(s4,p)` returns A_4 for every call. In order to call `FNormalizerWrtFormation` we must also set the property `IsIntegrated` to `true`.

```
gap> preform := rec( name := "ForComplement",
> fScreen := function( H, p )
> return Subgroup( H, GeneratorsOfGroup( H ){[2,3,4]});
> end);;
gap> form := Formation(preform);
formation of ForComplement groups
gap> SetIsIntegrated(form, true);
```

Now we may use `FNormalizerWrtFormation` with `s4` to get the complement, an S_3. (Recall that unless `form` already thinks it's integrated, `FNormalizerWrtFormation` will automatically integrate `form` before running its computations, which may not be the desired behavior.)

```
gap> comp := FNormalizerWrtFormation(s4, form); Size(comp);
Group([ f1, f2 ])
6
```

A user who wanted to employ the $\mathscr{F}$-normalizer technique to compute very many complements in this way would probably wish to create a new GAP function by extracting portions of the code that computes $\mathscr{F}$-systems.

Bibliography