The new `GroupTheory` package in Maple 17

Maple [1] is a well-known computer algebra package, initially developed in the early 1980s by the Symbolic Computation Group at the University of Waterloo. Maple has had a package dealing with group theory dating back to the first decade of its existence. It is called `group` and has been showing its limitations for a long time. For Maple 17, a completely new package called `GroupTheory` was added that replaces the older package. It was written at Maplesoft, incorporating many ideas and a substantial amount of code and data from two earlier packages written by the Computer Algebra Group at Simon Fraser University. One is a library of group presentations and permutation representations, written by Vahid Dabbaghian. The other package [2] focuses on visualization (it contained most of the code described in Section 4) and also performs substantial computation; it was written by Asif Zaman and Michael Monagan.

Other notable software packages dealing with discrete groups are GAP [3], Magma [4], and Mathematica [5]'s group theory features. We believe that, while some features offered by these packages are similar, each of the Sections 2, 3, and 4 describes some functionality that is unique to Maple.

1. **Basic functionality**

Groups can be represented in one of five ways: by permutations, by generators and relations, by an explicit multiplication table, as a set with manually defined group operations, and as an abstract group where elements cannot be listed but only properties computed. (This last option is explored in Section 2.) There is a substantial library of groups, containing all small groups of up to 200 elements (numbered consistently with the small group databases in GAP and Magma), all simple groups, many linear groups, and many individually named groups. These groups can typically be constructed in multiple representations: for example, using the `DihedralGroup` command, one can construct either the permutation or the finitely presented representation.

Such a group can then be interrogated about its properties, such as its order, whether it is simple, or its Fitting subgroup. Isomorphism testing between arbitrary groups is also supported. In total, the package contains about 120 commands for constructing and interrogating groups.

2. **Symbolic groups**

The `GroupTheory` package can deal with groups where it cannot list any elements explicitly. There are two such classes of groups: those where the group is defined only up to a symbolic parameter (such as `DihedralGroup(n)` or $\mathbf{D}_n$, the dihedral group with $2n$ elements), and those where listing elements is merely highly impractical, such as the Monster group $\mathbb{M}$.

For groups of the latter kind, Maple simply has many properties stored explicitly. Here are a few examples concerning the Monster group:

```
> GroupOrder(Monster());
```

$$808017424794512875886459904961710757005754368000000000$$

```
> IsSimple(DirectProduct(Monster(), TrivialGroup()));
```

$$\mathit{true}$$

```
> IsSimple(DirectProduct(Monster(), CyclicGroup(2)));
```

$$\mathit{false}$$

For groups defined with a symbolic parameter, Maple has some predefined properties as for the purely symbolic groups, but it can also deduce some properties from assumptions made on the parameters:

```
> IsNilpotent(DihedralGroup(6*n)) assuming n :: posint;
```

$$\mathit{true}$$

3. Interface

The `GroupTheory` package attempts to accept input and generate output in typography that is as close as possible to what one would traditionally find it in a textbook.

The requirements for input are clearly much stricter than for output, in that the input has to be processed by the parser for the general Maple language. Nonetheless, the package understands as input finitely presented groups in the format

$$\langle g_1, \ldots, g_n | r_1, \ldots, r_k \rangle,$$

where $g_1, \ldots, g_n$ are the generators and $r_1, \ldots, r_k$ are the relations.

For output, there is much more functionality. A few examples follow.

Input and output	Discussion
`> DirectProduct(Monster(), Symm(n));` $\mathbb{M} \times \mathbf{S}_n$	The direct product of the monster group and the symmetric group of order n.
`> DerivedSeries(Symm(4));` $\mathbf{S}_4 \triangleright \langle (1,3,2),(2,4,3) \rangle \triangleright \langle (1,3)(2,4),(1,4)(2,3) \rangle \triangleright \langle \rangle$	The derived series of the symmetric group of order 4.
`> g := PSL(2, 3);` $g := \mathbf{PSL}(2,3)$	
`> h := SylowSubgroup(3, g);` $h := \langle \text{a permutation group on 4 letters} \rangle$	The normalizer in $\mathbf{PSL}(2,3)$ of a Sylow-3 subgroup, h. Note that the generators of h are computed lazily, but once they are computed (in order to compute the order of h), they are remembered. This is discussed in Section 5.
`> k := Normaliser(h, g);` $k := N_{\mathbf{PSL}(2,3)}(\langle \text{a permutation group on 4 letters} \rangle)$	
`> GroupOrder(h);` 3	
`> k;` $N_{\mathbf{PSL}(2,3)}(\langle (2,3,4) \rangle)$	

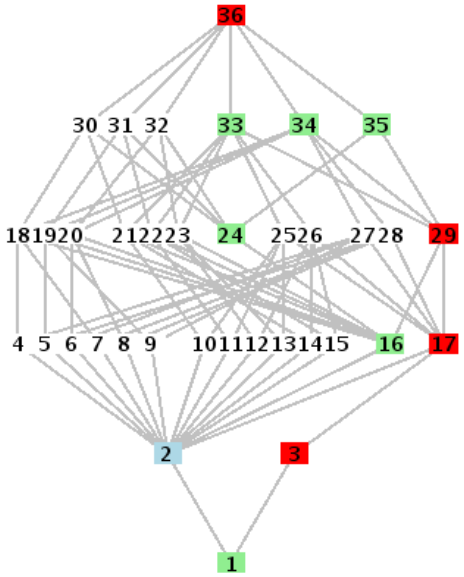

FIGURE 1. The subgroup lattice of the small group (48, 8). Highlighted in light blue is the centre, in light green are the normal subgroups, and in red is the lower central series.

4. Visualization

The `GroupTheory` package supports two visualizations: a subgroup lattice and a Cayley table.

An example of the subgroup lattice visualization can be found in Figure 1. This figure was obtained with the commands

```
> g := SmallGroup(48, 8):
> DrawSubgroupLattice(g, highlight=LowerCentralSeries(g));
```

Conjugate subgroups are placed next to each other, a little closer together than non-conjugate subgroups. This already shows, for example, what the normal subgroups are: the groups of size 1. For extra emphasis, any subset of subgroups can be highlighted in any combination of colours; the default is the centre and all normal subgroups. In the example, the lower central series is additionally highlighted (overriding the colours for the normal subgroups).

Two examples of the Cayley table visualization are shown in Figures 2 and 3. In Figure 2, the centre of the group is indicated by a thick black line. It consists of two elements. In Figure 3, the elements are highlighted depending on the coset of a particular subgroup they occur in.

5. Performance: memoization and autocompiled code

Groups are represented by objects (a type of modules). This provides part of the dispatching logic that selects the appropriate piece of code when a property needs to be computed for a given tuple of arguments. Another part is a memoization feature: most properties that are defined on a single argument, such as group order or simplicity, are computed only when necessary, but are remembered. That is, once such a property is computed, it is stored in a hash table in the object, with the property name as the key, and subsequent computations take advantage of this by looking it up rather than recomputing. A demonstration of this functionality is contained in Section 3.

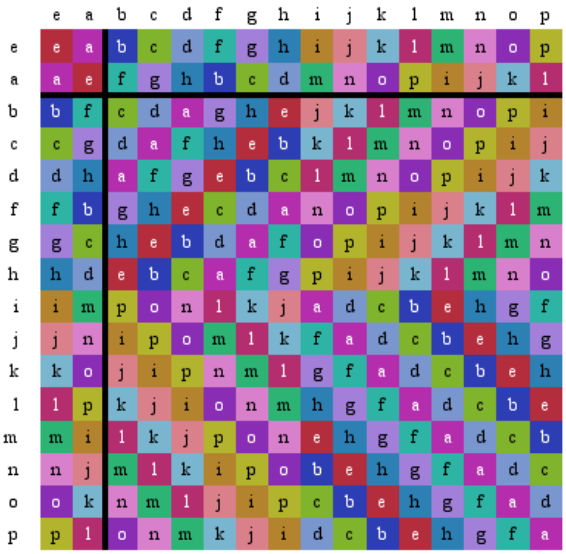

FIGURE 2. The Cayley table of the dicyclic group of order 16.

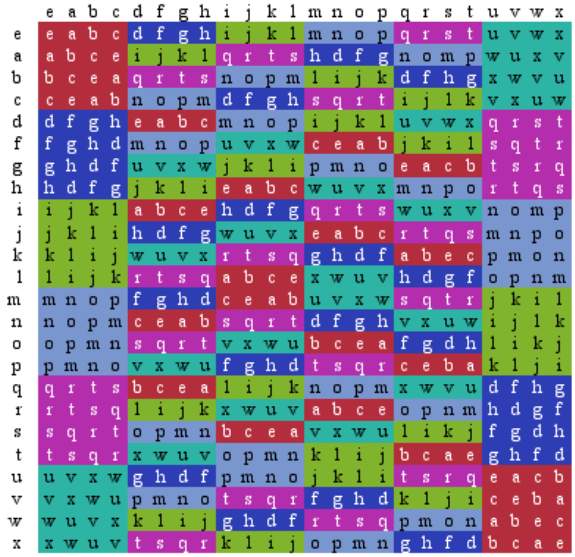

FIGURE 3. The Cayley table of a group of order 24.

For many of the low level algorithms, it was decided that the best tradeoff between performance and ease of programming was obtained by writing them in the subset of the Maple language that can be directly compiled into C code, and using the `autocompile` option. This means that the code is compiled on the fly.

As an example of these techniques, let us examine the `IdentifySmallGroup` command. It is used for identifying the isomorphism type of small groups (presently of size 200 or less) and uses the same numbering as its GAP and Magma equivalents. There are many shortcuts, for example if the group order n is prime. If these do not apply, the algorithm determines, for each possible order $d|n$, the numbers of group elements of order d, and additionally the number of elements of the derived subgroup. These numbers are determined from the Cayley table of the group by autocompilable code. It then evaluates a perfect hash function on this sequence of numbers to find all groups that share these characterstics. (The hash function value is one of the values remembered as described in Section 5.) Subsequently, as long as there is more than one candidate left, the code uses its general isomorphism testing command `AreIsomorphic`.

References

[1] *Maple 17*. Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario.

[2] Asif Zaman and Michael Monagan, *Visualizing Groups in Maple* (poster), `http://www.cecm.sfu.ca/research/posters/zaman09.pdf`.

[3] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.6.3*; 2013, (`http://www.gap-system.org`).

[4] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput., 24 (1997), 235265.

[5] Wolfram Research, Inc., *Mathematica, Version 9.0*, Champaign, IL (2012).