Interface to CARAT

Version 2.3.5

Franz Gähler

Fakultät für Mathematik, Universität Bielefeld

Postfach 10 01 31, D-33501 Bielefeld

gaehler@math.uni-bielefeld.de

Based on CARAT of 29 July 2022 at
https://github.com/lbfm-rwth/carat

by

J. Opgenorth, W. Plesken, T. Schulz

Lehrstuhl B für Mathematik, Templergraben 55, D-52056 Aachen

April 2023

Contents

1 Introduction

The package CaratInterface contains GAP interface routines to CARAT, a package of programs for the computation with crystallographic groups. CARAT is implemented in C, and has been developed by J. Opgenorth, W. Plesken, and T. Schulz at Lehrstuhl B für Mathematik, RWTH Aachen. The algorithms used by CARAT are described in [OPS98].

CARAT is to a large extent complementary to the GAP package Cryst. In particular, it provides routines for the computation of normalizers and conjugators of finite unimodular groups in $GL(n,Z)$, and routines for the computation of Bravais groups, which are all missing in Cryst. Furthermore, CARAT provides also a catalogue of Bravais groups up to dimension 6. Cryst automatically loads CARAT when it is available, and makes use of its functions where necessary. The present package thereby extends the functionality of Cryst considerably.

CARAT itself is NOT part of this package. However, for your convenience, and in accordance with the CARAT license, a copy of CARAT is included. It is this version with which the interface routines have been tested. The most recent version of CARAT can be obtained at its home page

`https://lbfm-rwth.github.io/carat` .

The GAP interface routines to CARAT have been written by

Franz Gähler
Fakultät für Mathematik
Universität Bielefeld
Postfach 10 01 31
D-33501 Bielefeld
`gaehler@math.uni-bielefeld.de`

For bug reports, suggestions and comments regarding the interface routines, please use the issue tracker on GitHub:

`https://github.com/gap-packages/CaratInterface/issues/`

Bug reports regarding CARAT itself should be reported in the CARAT issue tracker on GitHub:

`https://github.com/lbfm-rwth/carat/issues/`

2 Installation

GAP is distributed together with a large number of packages, including this one. CaratInterface is located inside the subdirectory `pkg/CaratInterface/` of your GAP installation, and the CARAT programs contained in it must be compiled. Compilation is most conveniently done with the script `BuildPackages.sh` from GAP. You can either build (almost) all packages, or only a single one by giving its name on the command line. Alternatively, you can compile CARAT inside CaratInterface manually, by executing these commands inside the directory `pkg/CaratInterface/`:

```
./configure <path-to-GAP-root>
make
```

The configure script optionally takes the path to the root directory of your GAP installation as argument. The default `../..` should usually work, if the package is located in the `pkg` subdirectory of the GAP root directory, but if you have unpacked CaratInterface to a location like `~/.gap/pkg/CaratInterface/`, you need to explicitly give the path to the GAP root directory as argument. The result of `configure` is written to the file `config.carat`, which can be inspected if something goes wrong.

If GAP was configured to use a specific GMP library, or the GMP library bundled with GAP, then CARAT will try to use that same GMP library. Otherwise, the system GMP library in the default path is chosen. If you want to use another GMP library, you may add an argument `--with-gmp=path-to-gmp` to the above configure command, where the directory `path-to-gmp` must contain subdirectories `lib/` and `include/` with the GMP library and include files, respectively.

If you want or need to add further compile or link flags, you may prepend `CFLAGS="your flags"` to the configure command, or append it to the make command (one of these is enough), so that the complete build commands including all options are these:

```
[CFLAGS="<your flags>"] ./configure [<path-to-GAP-root>] [--with-gmp=<path-to-gmp>]
make [CFLAGS="<your flags>"]
```

As CARAT's catalog of Q-classes of unimodular groups is rather large, it is unpacked by default only up to dimension 5. If you also want to unpack the data for dimension 6, you can do this with the extra command (again inside directory `pkg/CaratInterface/`)

```
make qcat6
```

This adds another 150 Mb of data to the installation.

3 Interface to CARAT

The GAP interface to CARAT consists of two parts, low level interface routines to CARAT functions on the one hand, and comfortable high level GAP functions on the other hand. The high level functions, implemented in terms of the low level functions, provide actually methods for functions and operations declared in the GAP library.

Note that while (almost) all CARAT functions should be accessible from within GAP by the low level interface routines, high level interface routines are provided only for a small subset of the CARAT functions. Priority has been given to routines providing functionality that has previously not been available in GAP. Further high level interface routines may be added in the future.

3.1 Action from the left and from the right

In crystallography, the convention usually is that matrix groups act from the left on column vectors. This convention is adopted also in CARAT. The low level interface routines described below must respect this convention and provide CARAT with data in the expected format.

On the other hand, in GAP the convention is that all groups act from the right, in the case of matrix groups on row vectors. However, in order to make GAP accessible to crystallographers, functions that are important in crystallography and for which it matters which action is assumed, are provided in two variants, one for each convention. The high level routines currently provided by this package do not depend on which convention is assumed. This may change, however, when further high level routines are added in the future.

3.2 CARAT input and output files

CARAT routines read their input from one or several input files, and write the result to standard output. In order to use CARAT routines from within GAP, the input must be prepared in suitably formatted input files. A CARAT command is then executed with these input files, with standard output redirected to an output file, which is read back into GAP afterwards. This section describes routines interfacing with CARAT input and output files.

Working with CARAT requires many temporary files. When the CARAT package is loaded, a temporary directory is created, where one can put such files. The routine

1 ▶ `CaratTmpFile(` *filename* `)` F

returns a file name *filename* in the CARAT temporary directory, which can be used to store temporary data. Of course, it is also possible to use any other file name, for instance files in the current directory.

2 ▶ `CaratShowFile(` *filename* `)` F

displays the contents of any text file on the terminal. This can be used to inspect the contents of CARAT input and output files.

Most CARAT data files are in either of two formats. The first CARAT file type is the Matrix File, containing one or several matrices. The following routines serve as interface to CARAT Matrix Files.

3 ▶ `CaratWriteMatrixFile(` *filename*`,` *data* `)` F

takes a file name and a matrix or a list of matrices, and writes the matrix or matrices to the file.

4 ► `CaratReadMatrixFile( filename )` F

reads a CARAT matrix file, and returns a matrix or a list of matrices read from the file.

The second CARAT file type is the Bravais File, containing information on a finite unimodular group. In GAP, the contents of a Bravais File is represented by a Bravais record, having the following components:

`generators`
: generators of the finite unimodular group

`formspace`
: basis of the space of invariant forms (optional)

`centerings`
: list of centering matrices (optional)

`normalizer`
: additional generators of the normalizer in GL(n,Z) (optional)

`centralizer`
: additional generators of the centralizer in GL(n,Z) (optional)

`size`
: size of the unimodular group (optional)

The following routines serve as interface to CARAT Bravais Files.

5 ► `CaratWriteBravaisFile( filename, data )` F

takes a file name and a Bravais record, and writes the data in the Bravais record to the file.

6 ► `CaratReadBravaisFile( filename )` F

reads a Bravais File, and returns the resulting Bravais record.

Certain CARAT programs produce output files containing several Bravais records, possibly preceeded by a varying number of header lines.

7 ► `CaratReadMultiBravaisFile( filename )` F

reads such a multi-Bravais file, and returns a record with the components `info` and `groups`. `info` is the list of header lines before the first Bravais record starts, and `groups` is the list of Bravais records read from the file.

3.3 Executing CARAT commands

To execute a CARAT program from within GAP, some low level, general purpose routines are provided in this package. Higher level routines for certain CARAT functions may be available in the GAP library or in other packages. These higher level functions are expected to use the following low level routines, so that changes in the low level interface will be transparent.

An arbitrary CARAT program can be executed with the routine

1 ► `CaratCommand( command, args, outfile )` F

where *command* is the name of a CARAT program, *args* is a string containing the command line arguments of the CARAT program, and *outfile* is the name of the file to which the output is to be written. Example:

```
gap> CaratCommand( "Z_equiv", "file1 file2", "file.out" );
```

A short description of the arguments and options of any CARAT program can be obtained from the CARAT online help facility with

2 ► `CaratHelp( command )` F

where *command* is the name of the CARAT program. CaratHelp executes the program with the `-h` option, and writes the output to the terminal. Example:

```
gap> CaratHelp( "Z_equiv" );
```

A list of all CARAT programs, along with a description of their usage and functionality, can be found in the CARAT documentation (in HTML), in the file

```
tex/introduction.html
```

in the CARAT home directory.

3.4 Methods provided by CARAT

CARAT implements methods for the following functions and operations declared in the GAP library. For a detailed description of these functions, please consult the GAP manual (section 44.6).

1 ► `BravaisGroup( G )` A

2 ► `IsBravaisGroup( G )` P

3 ► `BravaisSubgroups( G )` A

4 ► `BravaisSupergroups( G )` A

5 ► `NormalizerInGLnZBravaisGroup( G )` A

6 ► `Normalizer( GL(n, Integers), G )` O

7 ► `Centralizer( GL(n, Integers), G )` O

8 ► `ZClassRepsQClass( G )` A

9 ► `RepresentativeAction( GL(n,Integers), G1, G2 )` O

3.5 CARAT Bravais Catalog

CARAT contains a catalog with $\mathbb{Z}$-class representatives of all Bravais groups of dimension up to 6. These Bravais groups are accessed via a crystal family symbol.

1 ► `CaratCrystalFamilies[d]` V

returns a list of inequivalent crystal family symbols in dimension d.

2 ► `BravaisGroupsCrystalFamily( symb )` F

returns a list of $\mathbb{Z}$-class representatives of the Bravais groups in the crystal family with symbol *symb*.

3.6 CARAT Q-Class Catalog

CARAT contains a catalog with representatives of all Q-classes of finite unimodular groups up to dimension 6. This catalog can be accessd with the function

1 ▶ `CaratQClassCatalog( grp, mode )` F

where *grp* is a finite unimodular group of dimension up to 6, and *mode* is an integer. This function returns a record with one or several of the following components, depending on the decomposition of $mode = n_0 + n_1 * 2 + n_2 * 4$ into powers of 2:

`qclass`
: Q-class symbol; this component is always present

`familysymb`
: crystal family symbol (present if $n_0 <> 0$)

`trans`
: transformation to standard representative of Q-class: *grp^trans* = *std* (present if $n_1 <> 0$)

`group`
: standard representative of Q-class of *grp* (present if $n_2 <> 0$)

If *G1* and *G2* are two unimodular groups,

2 ▶ `ConjugatorQClass( G1, G2 )` F

returns a rational matrix *m* such that *G1^m* = *G2*, or fail, if the groups are not in the same Q-class. Since this function uses the CARAT Q-class catalog, only groups up to dimension 6 are supported. If this dimension is exceeded, an error is reported.

Bibliography

[OPS98] J. Opgenorth, W. Plesken, and T. Schulz. Crystallographic Algorithms and Tables. *Acta Cryst A* 54, 517–531 (1998).

Index

This index covers only this manual. A page number in *italics* refers to a whole section which is devoted to the indexed subject. Keywords are sorted with case and spaces ignored, e.g., "`PermutationCharacter`" comes before "permutation group".