SymPol Manual

Thomas Rehn and Achill Schürmann

Contents

1. Overview

SymPol computes restricted automorphisms of polyhedra and performs polyhedral description conversion up to a given or computed symmetry group.

This document will give you a step-by-step introduction to the usage of SymPol. You can find a very compact quick start guide in Section A on page 7.

2. Compile and install

2.1. Software requirements

SymPol comes already bundled with patched versions of [cdd] and [lrs] to actually perform polyhedral representation conversion. It also contains a copy of [PermLib] for computations with permutations. To use and compile SymPol the following external software is required:

Parts of the [Boost] library are required by both PermLib and SymPol. Boost has to be installed in version 1.34.1 or higher with its `program_options` and `serialization` libraries. `cdd`, `lrs` and SymPol also make use of the arbitrary precision arithmetics library [GMP], both in its C and C++ version. Building SymPol (see also the next section) is most easily accomplished with the [CMake] configuration system.

So on a Debian/Ubuntu-based computer you would install the following packages:

- `libboost-dev`
- `libboost-program-options-dev`
- `libboost-serialization-dev`
- `libgmp3-dev`
- `libgmpxx4ldbl` (GMP C++ interface)
- `cmake`

2.2. Building SymPol

2.2.1. Compiling and installing

If CMake, Boost and GMP are installed, SymPol can be compiled as follows:

```
~/sympol$ mkdir build && cd build
~/sympol/build$ cmake -DCMAKE_BUILD_TYPE=Release  ..
~/sympol/build$ make
```

You can then use SymPol directly from the build directory and, for instance, print the command line help with

```
~/sympol/build$ ./sympol/sympol -h
SymPol v0.1 and PermLib 0.2 with lrs 4.2c and cddlib 0.94f
Allowed options:
  -h [ --help ]                        produce help message
 [...]
```

You can also build SymPol from any other directory by calling

```
 cmake -DCMAKE_BUILD_TYPE=Release /path/to/sympol-source
```

If you want to install SymPol you can call `make install` as root. This will install SymPol into `/usr/local`. To choose a different prefix add an additional argument `-DCMAKE_INSTALL_PREFIX=/your/prefix/here` after the `-DCMAKE_BUILD_TYPE=Release`. On Linux systems you may have to call `ldconfig` as root afterwards so that the system knows about the new shared libraries (`cddgmp` and `lrsgmp`).

Assuming that the default installation directory is in your `$PATH`, you can check that SymPol is correctly installed and view the command line help by issuing

```
 $ sympol -h
```

2.2.2. Optional libraries

SymPol can use [`nauty`] and [`NTL`] for a possibly faster computation of polyhedral symmetries, but for several reasons these are not bundled with SymPol but have to be compiled and installed separately (see section 4). Assuming that you have downloaded and compiled both `nauty` and `NTL`, the following is required for use with SymPol.

- Copy `libntl.a` from the `NTL-src`-directory and the complete `include`-directory into `external/NTL` of SymPol.
- Please note that `nauty` is not open source and imposes usage restrictions. First you have to create a static library by `ar rcs libnauty.a naututil.o nauty.o nautil.o nautinv.o naugraph.o rng.o`. Then copy `libnauty.a` and `nauty.h` into `external/nauty` of SymPol.

If both `NTL` and `nauty` libraries are installed like this, run `cmake` from the build directory as described above.

3. Input format

SymPol mainly builds on the .ine/.ext file format as established by [`cdd`] and [`lrs`]. However, SymPol imposes one restriction and offers one extension to the format.

Regardless of the file format, the input filename has to be specified by the command-line argument `-i <filename>`. In most cases the `-i` can be omitted and ending the command with a filename suffices. *command-line*

3.1. H-representation

A polyhedron in H-representation, i.e. given by m inequalities in $n-1$ variables, can be represented in an .ine file with the following format

```
H-representation
begin
m n rational
 { list of inequalities }
end
```

Every inequality is expected to be in the form $a_0 + a_1x_1 + a_2x_2 + \cdots + a_{n-1}x_{n-1} \geq 0$. Such an inequality then is denoted in the .ine file as line $a_0\ a_1\ a_2\ \ldots\ a_{n-1}$. So, for instance, a two-dimensional triangle defined by

$$\begin{array}{ccccc} x_1 & & & \geq & 0 \\ & & x_2 & \geq & 0 \\ x_1 & + & x_2 & \leq & 1 \end{array}$$

can be represented by the file

```
* 2-dim triangle
H-representation
begin
3 3 rational
 0  1  0
 0  0  1
 1 -1 -1
end
```

Here, the first line denotes a comment, introduced by a starting '*' character.

3.2. V-representation

Similarly, if a polytope is given by m rays and vertices in dimension $n-1$, it can be represented by an .ext file according to

```
V-representation
begin
m n rational
 { list of vertices }
 { list of rays }
end
```

Every vertex v gets a line 1 v_1 v_2 ... v_{n-1} and every ray r a line 0 r_1 r_2 ... r_{n-1}.

3.3. Differences to cdd and lrs

As extension to this basic format, SymPol allows to include the automorphism group of the polyhedron, or parts of it, into the file. The user may specify after the `end` of the H- or V-representation a permutation group section as follows:

```
...
end
permutation group
p
 { list of #p group generators }
q
 { #q base points separated by whitespace }
```

The p group generators are to be given in cycle form, where commas separate cycles. The value q may be set to zero if no base of the group is known. A definition of a group base and what it is good for is explained in [Ser03], [HEO05] or [Reh10]. So to denote a group $G = \langle (1\,3)(4\,5), (2\,6\,5) \rangle$ with two generators and no base one would write:

```
...
end
permutation group
2
 1 3,4 5
 2 6 5
0
```

If you want to use $1, 2, 4, 5$ as a (potentially partial) base the section should look like

```
...
end
permutation group
2
 1 3,4 5
 2 6 5
4
 1 2 4 5
```

In contrast to `cdd` and `lrs`, in SymPol every inequality (H-representation case) or vertex and ray (V-representation case) has to be in exactly one line. At least `lrs` tolerates parts of an inequality or vertex spread over multiple lines, which SymPol currently does not support. However, SymPol comes with a Perl script that converts an .ine or .ext file into a suitable format (see Section B).

4. Computing restricted automorphisms

If no or only a few automorphisms are known a priori, the user has the possibility to compute *restricted automorphisms* of a polyhedron. These automorphisms may not be the full (combinatorial) symmetry group of the polyhedron, but it can be computed without full knowledge of both descriptions. We refer to [BDS09] for further details. SymPol offers two different implementations. One is a straight-forward implementation using a standard matrix inversion algorithm [CLRS09, Ch. 28] and PermLib to compute matrix automorphisms [Reh, Reh10].

The other one relies on [NTL] for matrix inversion over integers and [nauty] for graph automorphisms. This approach is usually a bit faster, but is included only as an compile-time option for two reasons: `nauty` is not released under a proper open source license, so it cannot be distributed with SymPol. The `NTL` library is GPL licensed but not easy to integrate into SymPol automatically. Section 2.2.2 contains a description for the advanced user of how to activate this implementation.

To just compute and print the restricted automorphism group of a polyhedron use the command-line switch `--automorphisms-only`. *command-line*

5. Description conversion

In order to perform a description conversion of a polyhedron SymPol offers several algorithms. One of the following has to be chosen at the command-line as an automatic strategy selection currently is work in progress.

5.1. Direct conversion

The most straightforward way is to compute the complete complementary description and filter up to symmetries afterwards. The user can choose between [`lrs`] and [`cdd`] to perform the description conversion task. One may also estimate the difficulty of a problem by using the estimation feature of `lrs`.

The command-line switch `-e` enables the estimation mode. In this mode only a difficulty estimation is made by `lrs` and the program exits. To perform a polyhedral representation conversion up to symmetry for "easy" polyhedra you can use the `-d` switch for direct conversion. What "easy" means is hard to specify but experiments so far suggest that polyhedra with an estimation of 40 or below are good for the direct conversion technique. More difficult problems should be treated with one of the recursive methods shown below. Note that the absolute value of the estimation depends on the speed of the computer on which the estimation is performed, so these numbers are not necessarily comparable between different machines. *command-line*

You also can choose between `cdd` and `lrs` for the core polyhedral computations (difficulty estimation so far only by `lrs`). Both programs may behave quite differently on the same input so it may be worthwhile to try both options for difficult problems. By default `lrs` is used. To replace it with `cdd` use the `--cdd` command-line switch. The interface to `cdd` is still a bit experimental and may not work in all cases. *command-line*

5.2. Recursive methods

More sophisticated algorithms are recursive Adjacency Decomposition Method (ADM) and Incidence Decomposition Method (IDM) [BDS09, Reh]. If the estimation of a problem exceeds 40 one of the following methods should be used.

The ADM is selected by the command-line argument `-a <threshold>`. For problems and arising subproblems whose estimation is below the given threshold the representation conversion is performed directly. Problems whose difficulty exceeds the threshold are broken into smaller subproblems with the Adjacency Decomposition Method. *command-line*

For some polyhedra it seems to be advantageous to also use the Incidence Decomposition Method in combination with ADM. The `--idm-adm <thresholdIDM> <thresholdADM>` command-line argument selects this strategy. This works like the pure ADM strategy before with an ADM threshold with one addition: Before a problem is estimated by `lrs` a heuristic IDM metric is computed. *command-line*

Problems with an IDM metric value below the given IDM threshold are treated with IDM. If they exceed this threshold `lrs` computes an estimation. Then based on the ADM threshold either ADM or direct conversion is applied. The IDM metric is still being developed but for the current implementation an IDM threshold of 10 is recommended. As mentioned above, for the ADM threshold a value of 40 seems reasonable.

Alternatively, the strategy can be made depending on the recursion level with the `--idm-adm-level <levelIDM> <levelADM>` command-line argument. This will use IDM for the first *levelIDM* levels (may be 0) and ADM for the first *levelADM*, if IDM is not used. After *levelADM* direct computation will be used. This strategy helps to avoid expensive difficulty estimations. *command-line*

5.3. Examples

Direct conversion

```
sympol -d -i input-file
```

ADM

```
sympol -a 40 -i input-file
```

IDM and ADM combined

```
sympol --idm-adm 10 40 -i input-file

# use ADM for the first two recursion levels
sympol --idm-adm-level 0 2 -i input-file
```

5.4. Memory usage and other parameters

The memory usage of SymPol is dominated by the number of orbit elements it is allowed to store in RAM. Storing orbits in RAM allows to decide fast whether a new vertex/ray is equivalent under symmetry to one computed before. If not all orbits can be stored a quite expensive calculation is started to check for equivalence [Reh10, Ch. 3]. Thus the memory limit for orbits, specified by the command-line argument `--conf-compute-orbit-limit <number>`, should be chosen as large as possible. The default value 1024 means that SymPol will pre-compute orbits as long as it occupies less than 1024 megabytes of RAM. If this memory limit is exceeded vertex/ray equivalence will be computed by other means. *command-line*

5.5. Combinatorial features

SymPol can compute the adjacency graph of the description conversion result, up to the used symmetries. Construction of the adjacency graph requires the use of the ADM at least at the first recursion level. In this case the command-line option `--adjacencies` activates the adjacency graph computation. The adjacency graph is printed in a format suitable for visualization with [`Graphviz`]. The vertex numbers correspond to the position of the vertex/facet in the output list above. *command-line*

Example

```
sympol --adjacencies --idm-adm-level 0 1 -i input-file
```

If you copy the adjacency part of the output into a textfile `adjacencies.dot` and have [`Graphviz`] installed you can generate, for instance, a PNG graphic `adjacencies.png` of the graph by

```
neato -Tpng -o adjacencies.png adjacencies.dot
```

6. Output format

The output format follows the .ine/.ext format. The only difference is that the data section contains only rays/inequalities up to symmetry. The computed automorphisms group and the base used are printed, following the description in Section 3.3.

7. Other program options

By default SymPol prints usage statistics about used processor time and RAM and warnings and errors. If you want a more verbose output you can specify the `-v` parameter, followed by a number which represents the logging level: `INFO` (1), `DEBUG` (2), `DEBUG2` through `DEBUG5` (3–6). *command-line*

The command-line switch `-t` enables time measurement and prints the CPU time used at the end of the computation.

A. Quick Start Guide

Install If [CMake], [Boost] and [GMP] are installed:

```
~/sympol$ mkdir build && cd build
~/sympol/build$ cmake -DCMAKE_BUILD_TYPE=Release  ..
~/sympol/build$ make
# as root or in su/sudo shell
~/sympol/build$ make install
```

Then the SymPol binary will be installed in `/usr/local/bin`. On Linux systems you may have to call `ldconfig` as root afterwards so that the system knows about the new shared libraries (`cddgmp` and `lrsgmp`).

Compute the restricted automorphism group

```
sympol --automorphisms-only input-file
```

Estimate the difficulty of a representation conversion

```
sympol -e input-file
```

Do a representation conversion For "easy" input (see estimation, probably estimation below 40), try:

```
sympol -d input-file
```

For "difficult" input, try

```
sympol -a 40 input-file
```

and experiment with the ADM threshold `-a` if necessary.

Compute the adjacency graph after conversion

```
sympol --idm-adm-level 0 1 --adjacencies -i input-file
```

If you copy the adjacency part of the output into a textfile `adjacencies.dot` and have [Graphviz] installed you can generate, for instance, a PNG graphic `adjacencies.png` of the graph by

```
neato -Tpng -o adjacencies.png adjacencies.dot
```

B. Directory overview

After you extract a SymPol distribution package, you will find the following directories:

- `contrib` contains scripts to manipulate .ine/.ext files.
- `data` contains various example polyhedra in .ine/.ext files.
- `external` contains all third-party software used by SymPol: [cdd], [lrs], [PermLib].
- `sympol` contains the source code of the SymPol application

C. License

As both `cdd` and `lrs` are published under GPL2, SymPol is also available under GPL2. The complete text of the license is available from `http://www.gnu.org/licenses/gpl-2.0.html`.

Parts of the source code come with a more liberal license: The author's PermLib is BSD licensed.

D. References

Literature

[BDS09] David Bremner, Mathieu Dutour Sikiric, and Achill Schürmann. Polyhedral representation conversion up to symmetries. In David Avis, David Bremner, and Antoine Deza, editors, *Polyhedral computation*, CRM Proceedings & Lecture Notes, pages 45–72. American Mathematical Society, 2009. Available from: `http://arxiv.org/abs/math/0702239`.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[HEO05] Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of Computational Group Theory*. Discrete Mathematics and Applications. Chapman & Hall/CRC, 2005.

[Reh] Thomas Rehn. Polyhedral description conversion up to symmetries: Theory and application. Diploma thesis (mathematics), Otto von Guericke University Magdeburg. In preparation.

[Reh10] Thomas Rehn. Fundamental Permutation Group Algorithms for Symmetry Computation. Diploma thesis (computer science), Otto von Guericke University Magdeburg, February 2010.

[Ser03] Ákos Seress. *Permutation Group Algorithms*. Cambridge University Press, 2003.

Software

`[Boost]` Boost free peer-reviewed portable C++ source libraries. `http://www.boost.org/`.

`[cdd]` cdd, cddplus and cddlib by K. Fukuda. `http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html`.

`[CMake]` CMake – Cross Platform Make. `http://www.cmake.org/`.

`[GMP]` GMP, The GNU Multiple Precision Arithmetic Library. `http://gmplib.org/`.

`[Graphviz]` Graphviz – Graph Visualization Software. `http://www.graphviz.org/`.

`[lrs]` lrs by D. Avis. `http://cgm.cs.mcgill.ca/~avis/C/lrs.html`.

`[nauty]` nauty, computing automorphism groups of graphs and digraphs, by B. McKay. `http://cs.anu.edu.au/~bdm/nauty/`.

`[NTL]` NTL: A Library for doing Number Theory by V. Shoup. `http://www.shoup.net/ntl/`.

`[PermLib]` PermLib, a C++ library for permutation computations, by T. Rehn. `http://fma2.math.uni-magdeburg.de/~latgeo/PermLib-0.1/HTML/permlib.html`.