Orbiter User's Guide

Anton Betten

February 7, 2019

Abstract

We discuss how to use the program system Orbiter for the classification of combinatorial objects.

1 Introduction

Orbiter [1] [2] is a software package for the classification of combinatorial objects, written in C++. Orbiter consists of a large library of objects related to combinatorics, algebra, group theory and geometry. A number of applications come bundled with Orbiter, many of which are concernde with the classification or combinatorial objects: optimal linear codes, cubic surfaces with 27 lines over a finite field, and other objects in geometry. There are applications related to graph theory, all the way from Cayley graphs to distance regular graphs to clique finding algorithms and to algorithms for the classification of small graphs. Many of the algorithms in Orbiter allow for parallel computing. At this point Orbiter does not have a user interface. As a substitute, the bundled applications can be invoked using command line arguments. These commands can be collected in shell scripts or makefiles. While this is perhaps a bit uncomfortable for the novice user, it has the advantage that the computations are documented. Also, the results are typically written to file, waiting to be processed by other programs. It is also possible to write new applications in Orbiter. After all, this is what a class library is supposed to be good for. For this, a slightly higher level of familiarity with the conventions and data structures in Orbiter is required. Documentation of Orbiter is available on a web site, created using Doxygen. Any Orbiter application is developed using the Orbiter library (liborbiter.a), and has to be linked with the C++ standard library (cf. Figure 1). The Orbiter library is layered. There are five separate namespaces, some of which building on top of the other. This is shown in Figure 2.

2 Orbiter and Group Theory

The classifiction of combinatorial objects requires a group whose orbits correspond to the objects under consideration. For this reason, Orbiter provides finite groups and their group actions. Certain groups are provided with a standard action. New actions can be built from these using various constructions. The groups in Orbiter fall into four categories:

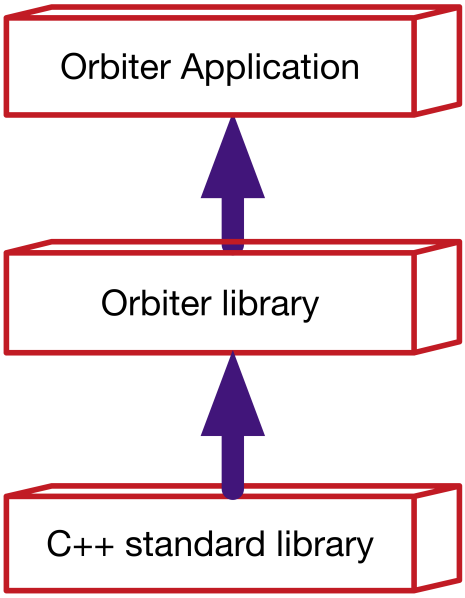

Figure 1: The orbiter model

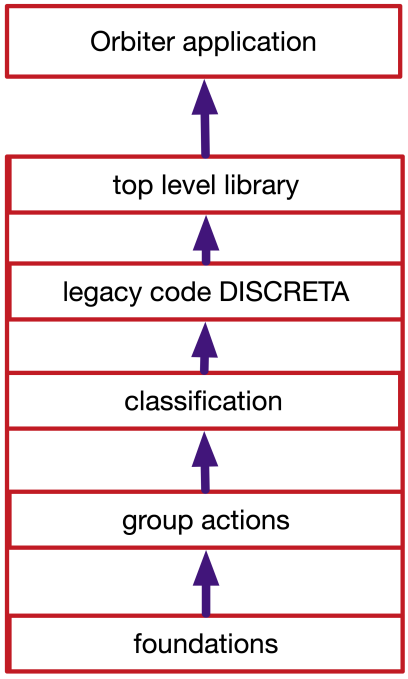

Figure 2: The namespaces of the orbiter library

Command	Arguments	Group
-GL	n, q	$\mathrm{GL}(n, q)$
-GGL	n, q	$\Gamma\mathrm{L}(n, q)$
-SL	n, q	$\mathrm{SL}(n, q)$
-SSL	n, q	$\Sigma\mathrm{L}(n, q)$
-PGL	n, q	$\mathrm{PGL}(n, q)$
-PGGL	n, q	$\mathrm{P}\Gamma\mathrm{L}(n, q)$
-PSL	n, q	$\mathrm{PSL}(n, q)$
-PSSL	n, q	$\mathrm{P}\Sigma\mathrm{L}(n, q)$
-AGL	n, q	$\mathrm{AGL}(n, q)$
-AGGL	n, q	$\mathrm{A}\Gamma\mathrm{L}(n, q)$
-ASL	n, q	$\mathrm{ASL}(n, q)$
-ASSL	n, q	$\mathrm{A}\Sigma\mathrm{L}(n, q)$

Table 1: Basic types of Orbiter matrix groups

(a) Matrix groups. These groups act linearly or semilinearly on a vector space. There are three types of group actions to be considered: projective groups, affine groups and general linear groups. Various other types exist, such as orthogonal groups and unitary groups. The elements of matrix groups are stored as matrices, possibly extended by a translation vector or a field automorphism.

(b) Permutation groups. These are groups of permutations of a finite set. The mappings are stored by listing each image, using a vector data structure.

(c) Direct product type.

(d) Wreath product type $\mathrm{GL}(n, q) \wr \mathrm{Sym}(n)$.

3 Matrix Groups

The command

`-linear`⟨arguments⟩⟨modifier⟩`-end`

can be used to select a matrix group. The arguments can be one of the commands in Table 1 (including two numerical values for n and q, respectively). The executable `linear_group.out` can be used to create a matrix group. For instance

```
linear_group.out -v 3 -linear -PGL 2 11 -end
```

creates $\mathrm{PGL}(2, 11)$ in the action on the 12 points of the projective line. The option `-v` $\langle k \rangle$ can be used to specify the verbosity of the command. Higher values of k lead to more text

Modifier	Arguments	Meaning
-wedge		action on the exterior square
-PGL2OnConic		induced action of $\mathrm{PGL}(2,q)$ on the conic in the plane $\mathrm{PG}(2,q)$
-monomial		subgroup of monomial matrices
-diagonal		subgroup of diagonal matrices
-null_polarity_group		null polarity group
-symplectic_group		symplectic group
-singer	k	subgroup of index k in the Singer cycle
-singer_and_frobenius	k	subgroup of index k in the Singer cycle, extended by the Frobenius automorphism of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$
-subfield_structure_action	s	action by field reduction to the subfield of index s
-subgroup_from_file	f l	read subgroup from file f and give it the label l
-borel_subgroup_upper		Borel subgroup of upper triangular matrices
-borel_subgroup_lower		Borel subgroup of lower triangular matrices
-identity_group		identity subgroup
-on_k_subspaces	k	induced action on k dimensional subspaces
-orthogonal	ϵ	induced action on the orthogonal geometry, with $\epsilon \in \{\pm 1\}$ when n is even

Table 2: Modifiers for creating matrix groups

output. A verbosity $k = 0$ means no or almost no text output. If the verbosity is positive, the program will print the chosen generators, as well as a list of points for the permutation representation.

The modifier can be any command from Table 2. For instance, the command

```
linear_group.out -v 3 -linear -PGL 3 11 -singer_and_frobenius 190 -end
```

creates a group of order 21 which is formed by taking the subgroup of order 7 in the Singer cycle of $\mathrm{PGL}(3,11)$ extended by the Frobenius automorphism of $\mathbb{F}_{11^3}$ over $\mathbb{F}_{11}$ of order 3. In order to create the subgroup of order 7, the generator of the Singer cycle is considered, of order $11^3 - 1 = 1330$. Since $1330 = 190 \cdot 7$, raising the generator to the power of 190 creates an element of order 7. The group generated acts on the $11^2 + 11 + 1 = 133$ points

of PG(2, 11). Thus, a matrix group of order 21 is generated with a permutation action of degree 133. The matrix generators are

$$\begin{bmatrix} 1 & 2 & 6 \\ 9 & 6 & 2 \\ 3 & 7 & 6 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 1 & 7 & 7 \\ 5 & 6 & 3 \end{bmatrix}.$$

The first matrix is the element of order 7 arising from the Singer cycle. The second matrix arises from the Frobenius automorphism.

4 Installing Orbiter

Orbiter is available on github (`https://github.com`). Search for "abetten/orbiter" or go directly to

`https://github.com/abetten/orbiter`

Once there, find the green button called "Clone or download". The button offers two options: "Open in Desktop" and "Download ZIP" Choose one of them to download Orbiter. Orbiter is compiled with makefiles. Some system specific comments are in order:

- For Microsoft Windows users, it is recommended to install Orbiter using `cygwin` [3]. Cygwin is a Unix-like environment and command-line interface for Microsoft Windows.
- Macintosh users need to install Xcode (search for xcode in the app store). Xcode is an integrated development environment (IDE) for MacOS. It comes with command line tools for software development. In order to compile Orbiter, the command line tools are required. There is no need to use the integrated development environment though.
- For Linux users, the compiler environment (for instance Gnu C++) needs to be installed.

We will use a terminal window (console) to install Orbiter. Assuming that we have the various compiler tools avalable, the installation proceeds as follows. The following commands are typed into the terminal window.

Enter the directory `ORBITER/src` and type

```
make
```

This should create a lot of text output to the console. Assuming that the command executes without errors, orbiter is now ready. The specific purpose of this make command is to compile all C++ source code into object files, to bind the object files together into one library, and to link the orbiter executables. The C++ cource code is in files with extension .cpp. There are additional filed called header files with extension .h. The header files are needed to compile

the .cpp files into .o files. This has to do that one source file needs to know a little bit what goes on in the other source files. The object files are corresponding files with extension .o.

$$\Big(\text{XXX.cpp, orbiter.h} \Big) \mapsto \text{XXX.o} \mapsto \text{liborbiter.a.}$$

Here, XXX stands for all files in the /src/lib subtree, and the map XXX.cpp $\mapsto$ XXX.o is one-to-one. The map XXX.o $\mapsto$ liborbiter.a is many-to-one. Once the library has been compiled, the application executables are compiled:

$$\left.\begin{array}{c} \text{YYY.cpp} \mapsto \text{YYY.o} \\ \text{liborbiter.a} \\ \text{standard libraries (libc++ etc.)} \end{array}\right\} \mapsto \text{YYY.out}$$

This time, the source files YYY reside in the src/apps branch. The file YYY.out is the executable. Executables are programs which can be called. They are recognizable by the x flag in the directory list). The executable contains the actual program to do the work. It will be called for instance through the command line. This make command has to be executed only once. One can recognize the fact that make has been successful by verifying the presence of files with extension .o on the src subdirectories. Also, various files with the extension .out have been created in the subdirectories of src/apps. The make command descends into all subdirectories of src and performs make.

In order to test orbiter, go to the subdirectory `ORBITER/examples`. There are several subdirectories containing test problems. The test problems are calls to orbiter, asking it so solve small problems. For instance, the subdirectory groups contains some problems related to groups. Once in the directory `ORBITER/examples/groups`, issue the command `make` to run the first problem. To see what kind of problems are available, open the file `makefile` and see what targets are defined. You can type `make target` where `target` is any of the targets in the makefile to run that particular problem. Most problems will create a lot of output to the console.

A Orbiter Class List

Here are the classes, structs, unions and interfaces with brief descriptions. In total, there are 214 classes and similar objects.

Table 3: Summary of Orbiter Classes.

Class	Purpose
a_domain	Related to the computation of Young representations
action	The class action implements a permutation group in a fixed action
action_by_conjugation	Action by conjugation on the elements of a given group
action_by_representation	Action of PSL(2,q) on a conic (the only type implemented so far)
action_by_restriction	Restricted action on an invariant subset
action_by_right_multiplication	Action on a the set of elements of a group by right multiplication
action_by_subfield_structure	Action on the vector space arising from a field over a subfield
action_is_minimal_data	Internal class for is_minimal backtracking used by class action
action_on_andre	Action on the elements of a projective plane constructed via Andre / Bruck / Bose
action_on_bricks	Related to a problem of Neil Sloane
action_on_cosets	Action on the cosets of a subgroup by right multiplication
action_on_determinant	Action on the determinant of a group of matrices (used to compute the subgroup PSL)
action_on_factor_space	Induced action on the factor space of a vector space modulo a subspace
action_on_flags	Action on flags
action_on_grassmannian	Action on the grassmannian (subspaces of a fixed dimension of a vectors space)
action_on_homogeneous_polynomials	Induced action on the set of homogeneous polynomials over a finite field
action_on_k_subsets	Induced action on k-subsets of a set of size n
action_on_orbits	Induced action on the set of orbits (usually by the normalizer)
action_on_orthogonal	Action on the orthogonal geometry

Continued on next page

Table 3 – continued from previous page

Class	Purpose
action_on_sets	Induced action on a given set of sets
action_on_sign	Action on the sign function of a permutation group (to compute the even subgroup)
action_on_spread_set	Induced action on a spread set via the associated spread
action_on_subgroups	Induced action on subgroups of a group
action_on_wedge_product	Wedge product action on exterior square of a vector space
andre_construction	Andre / Bruck / Bose construction of a translation plane from a spread
andre_construction_ line_element	Related to class andre_construction
andre_construction_ point_element	Related to class andre_construction
arc_generator	Poset classification for arcs in desarguesian projective planes
arc_lifting	Creates a cubic surface from a 6-arc in a plane
blt_set	Classification of BLT-sets
BLT_set_create	To create a BLT-set from a known construction
BLT_set_create_description	To describe a BLT set with a known construction from the command line
brick_domain	For a problem of Neil Sloane
bt_key	DISCRETA class for databases
btree	DISCRETA class for a database
btree_page_registry_key_pair	DISCRETA internal class related to class database
buekenhout_metz	Buekenhout Metz unitals
buffer	DISCRETA auxiliary class related to the class database
cayley_graph_search	For a problem of Ferdinand Ihringer
choose_points_or_lines	To classify objects in projective planes
classification	Poset classification data structure
classify	Statistical analysis of vectors of ints
classify_bitvectors	Stores the canonical form of 0/1 matrices for the purposes of classification
classify_double_sixes	To classify double sixes in PG(3,q)
classify_trihedral_pairs	To classify double triplets in PG(3,q)
clique_finder	A class that can be used to find cliques in graphs

Continued on next page

Table 3 – continued from previous page

Class	Purpose
code_generator	Classification of optimal linear codes over Fq
colored_graph	Graph with a vertex coloring
compute_stabilizer	Wrapper to compute the set stabilizer using the poset classification algorithm
coset_table_entry	Helper class for the poset classification algorithm
data_file	To read files of classifications from the poset classification algorithm
database	DISCRETA class for a database
datatype	DISCRETA auxiliary class related to the class database
decomposition	Decomposition of an incidence matrix
desarguesian_spread	Desarguesian spread
design_data	DISCRETA class for Kramer Mesner type problems
design_parameter	DISCRETA class for design parameters
design_parameter_source	DISCRETA class for the design parameters database
difference_set_in_heisenberg_group	Class related to a problem of Tao
diophant	For diophantine systems of equations (i.e., linear systems over the integers)
direct_product	Direct product of two matrix groups in product action
direct_product_action	Searching for line transitive point imprimitive designs preserving a grid structure on points
discreta_base	DISCRETA base class. All DISCRETA classes are derived from this class
dlx_node	Internal class for the dancing links exact cover algorithm
domain	DISCRETA class for influencing arithmetic operations
eckardt_point	Eckardt point on a cubic surface using the Schlaefli labeling
elliptic_curve	Fixed elliptic curve in Weierstrass form
exact_cover	Wrapper class for exact cover problems arising with the lifting of combinatorial objects
exact_cover_arguments	Command line arguments to control the lifting via exact cover
extension	Class representing a flag orbit
factor_group	Auxiliary class for create_factor_group, which is used in analyze_group.cpp
fancy_set	To store a subset of size k of a set of size n

Continued on next page

Table 3 – continued from previous page

Class	Purpose
ff_memory	DISCRETA auxilliary class for class domain
file_output	Wrapper class for an ofstream which allows to store extra data
finite_field	Finite field Fq
finite_ring	Finite chain rings
flag	Maximal chain of subspaces
flag_orbit_node	Related to the class flag_orbits
flag_orbits	Related to the class classification
generators_symplectic_group	Auxilliary class to construct generators of the symplectic group
geo_parameter	Input parameters for TDO-process
geometry	DISCRETA class for incidence matrices
gl_class_rep	To represent a conjugacy class of matrices in GL(n,q)
gl_classes	Conjugacy classes in GL(n,q)
graph_generator	Classification of graphs and tournaments
graph_layer	Part of the data structure layered_graph
graph_node	Part of the data structure layered_graph
grassmann	To rank and unrank subspaces of a fixed dimension in $\mathbb{F}_q^n$
grassmann_embedded	Subspaces with a fixed embedding
grid_frame	Class to help with drawing elements in a 2D grid fashion
group	Container data structure for groups
group_selection	DISCRETA class to choose a group from the command line or from a UI
hadamard	Classification of Hadamard matrices
heisenberg	The Heisenberg group of n x n matrices
hermitian	Hermitian space
hjelmslev	Hjelmslev geometry
hollerith	DISCRETA string class
homogeneous_polynomial_domain	Homogeneous polynomials in n variables over a finite field
incidence_structure	Incidence structure interface for many different types of geometries
int_matrix	Class to represent matrices over int
int_vector	Vector on ints

Continued on next page

Table 3 – continued from previous page

Class	Purpose
integer	DISCRETA integer class
isomorph	Hybrid algorithm to classify combinatorial bjects
isomorph_arguments	Helper class for isomorph
isomorph_worker_data	Auxiliary class to pass case specific data to the function isomorph_worker
itemtyp	DISCRETA auxiliary class related to the class database
job_table	Job table for the scheduler app for parallel processing
k_arc_generator	To classify k-arcs in the projective plane PG(2,q)
keycarrier	DISCRETA auxiliary class related to the class database
klein_correspondence	Klein correspondence between lines in PG(3,q) and points on the Klein quadric
knarr	Knarr construction of a GQ from a BLT-set
layered_graph	Data structure to store partially ordered sets
layered_graph_draw_options	Options for drawing an object of type layered_graph
linear_group	Used to create a linear group from command line arguments
linear_group_description	Description of a linear group from the command line
linear_set	Classification of linear sets
longinteger	DISCRETA class for integers of arbitrary magnitude
longinteger_domain	Domain to compute with objects of type longinteger
longinteger_object	Class to represent aritrary precision integers
longinteger_representation	DISCRETA internal class to represent long integers
matrix	DISCRETA matrix class
matrix_access	DISCRETA utility class for matrix access
matrix_block_data	Auxiliary class for conjugacy classes in GL(n,q)
matrix_group	A linear group implemented as matrices
mem_object_registry	Maintains a registry of allocated memory
mem_object_registry_entry	Class related to mem_object_registry
memory	DISCRETA class to serialize data structures
memory_object	Can be used for serialization
mindist	Internal class for the algorithm to compute the minimum distance of a linear code
mp_graphics	General 2D graphical output interface (metapost, tikz, postscript)
norm_tables	Tables for the norm map in a finite field

Continued on next page

Table 3 – continued from previous page

Class	Purpose
null_polarity_generator	Internal class to compute generators for the group of null polarities
number_partition	DISCRETA class for partitions of an integer
object_in_projective_space	Geometric object in projective space (points, lines or packings)
object_in_projective_space_with_action	To represent an object in projective space
OBJECTSELF	DISCRETA internal class
orbit_node	Related to the class classification
orbit_of_equations	Schreier tree for action on homogeneous equations
orbit_of_sets	Schreier tree for action on subsets
orbit_of_subspaces	Schreier tree for action on subspaces
orbit_rep	To hold one orbit after reading files from Orbiters poset classification
orbit_transversal	Container data structure for a poset classification from Orbiter output
orbiter_data_file	Class to read output files from orbiters poset classification
orthogonal	Orthogonal geometry $O^\epsilon(n, q)$
ovoid_generator	Classification of ovoids in orthogonal spaces
page_storage	Data structure to store group elements in compressed form
page_table	DISCRETA class for bulk storage
pagetyp	DISCRETA auxiliary class related to the class database
partitionstack	Leon type partitionstack class
perm_group	An abstract permutation group
permutation	DISCRETA permutation class
plane_data	Auxiliary class for the class point_line
point_line	Data structure for general projective planes, including nodesarguesian ones
polar	Orthogonal geometry as a polar space
poset_classification	Poset classification algorithm (Snakes and Ladders)
poset_orbit_node	Class representing one poset orbit, related to the class poset_classification
printing_mode	DISCRETA class related to printing of objects
product_action	Product action of two group actions

Continued on next page

Table 3 – continued from previous page

Class	Purpose
projective_space	Projective space PG(n,q) of dimension n over Fq
projective_space_with_action	Projective space PG(n,q) with automorphism group PGGL(n+1,q)
rainbow_cliques	To search for rainbow cliques in graphs
rank_checker	Used to check that any d-1 columns are linearly independent
recoordinatize	Utility class to classify spreads
regular_ls_generator	Classification of regular linear spaces
representatives	Auxiliary class for class isomorph
scene	Collection of 3D geometry objects
schreier	Schreier trees for orbits on points
schreier_sims	Schreier Sims algorithm
search_blocking_set	To classify blocking sets in projective planes
set_and_stabilizer	Set and its known set stabilizer
set_of_sets	To store a set of sets
set_stabilizer_compute	Wrapper to compute the set stabilizer with the class compute_stabilizer
sims	A stabilizer chain for a permutation group
singer_cycle	Singer cycle in PG(n-1,q)
six_arcs_not_on_a_conic	To classify six-arcs not on a conic in PG(2,q)
solid	DISCRETA class for polyhedra
solution_file_data	Internal class related to tdo_data
spread	To classify spreads of PG(k-1,q) in PG(n-1,q) where n=2*k
spread_create	To create a known spread
spread_create_description	To describe the construction of a known spread from the command line
spread_lifting	Create spreads from smaller spreads
spreadsheet	For reading and writing of csv files
strong_generators	Strong generating set for a permutation group with group order
subfield_structure	Finite field as a vector space over a subfield
subgroup	List a subgroup of a group by storing the element indices
subspace_orbits	Poset classification for orbits on subspaces
surface	Cubic surfaces in PG(3,q) with 27 lines

Continued on next page

Table 3 – continued from previous page

Class	Purpose
surface_classify_wedge	To classify cubic surfaces using double sixes as substructures
surface_create	To create a cubic surface from a known construction
surface_create_description	To describe a known construction of a cubic surface from the command line
surface_object	Particular cubic surface in PG(3,q), given by its equation
surface_object_with_action	Instance of a cubic surface together with its stabilizer
surface_with_action	Cubic surfaces in projective space with automorphism group
symmetry_group	Internal class related to action
tdo_data	Class related to the class tdo_scheme
tdo_parameter_calculation	Main class for tdo_refine to refine the parameters of a linear space
tdo_scheme	TDO scheme captures the combinatorics of a decomposed incidence structure
tensor_product	Class for the app wreath_product
translation_plane_via_andre_model	Translation plane created via Andre / Bruck / Bose
tree	Data structure for trees
tree_node	Part of the data structure tree
union_find	Union find data structure (used in the poset classification)
union_find_on_k_subsets	Union find data structure (used in the poset classification)
unipoly	DISCRETA class for poynomials in one variable
unipoly_domain	Domain to compute with polynomials in one variable over a finite field
unusual_model	Penttila's unusual model to create BLT-sets
upstep_work	Helper class for the poset classification algorithm
Vector	DISCRETA vector class for vectors of DISCRETA objects
vector_ge	Vector of group elements
vector_hashing	Hash tables
W3q	W(3,q) generalized quadrangle
with	DISCRETA class related to class domain

Continued on next page

Table 3 – continued from previous page

Class	Purpose
wreath_product	Wreath product group AGL(d,q) wreath Sym(n)
young	The Young representations of the symmetric group

B Ovoid search in $O^-(6,2)$

Here is the makefile for the ovoid search in $O^-(6,2)$:

```
1       MY_PATH=~/DEV.18
2       SRC=$(MY_PATH)/GITHUB/orbiter/ORBITER/SRC
3       SRC2=$(MY_PATH)/ORBITER2/SRC2
4
5       OVOID_PATH=$(SRC)/APPS/OVOID
6       TOOLS_PATH=$(SRC)/APPS/TOOLS
7
8
9       Op42:
10         $(OVOID_PATH)/ovoid.out -v 2 -epsilon 1 -d 4 -q 2
11
12      Om42:
13         $(OVOID_PATH)/ovoid.out -v 2 -epsilon -1 -d 4 -q 2
14
15      Op62:
16         $(OVOID_PATH)/ovoid.out -v 2 -epsilon 1 -d 6 -q 2
17
18      Om62:
19         $(OVOID_PATH)/ovoid.out -v 5 -epsilon -1 -n 5 -q 2 -draw_poset -embedded -W
20
21      # order 25920, degree 27 = U_4(2) = Weyl group of type E_6 = Sp_4(2) = O_5(3)
22
23
24      draw:
25         $(TOOLS_PATH)/layered_graph_main.out -v 4 \
26            -file ovoid_-1_6_2_poset_lvl_9.layered_graph \
27            -draw test \
28            -rad 25000 \
29            -xin 1000000 \
30            -yin 1000000 \
31            -xout 1000000 \
32            -yout 1000000 \
33            -embedded \
34            -scale .44 \
35            -line_width 1.0 \
36            -y_stretch 0.8
37         pdflatex ovoid_-1_6_2_poset_lvl_9_draw.tex
38         open ovoid_-1_6_2_poset_lvl_9_draw.pdf
39
40
```

C The class action

In Figur 3, the collaboration graph for the class action is shown.

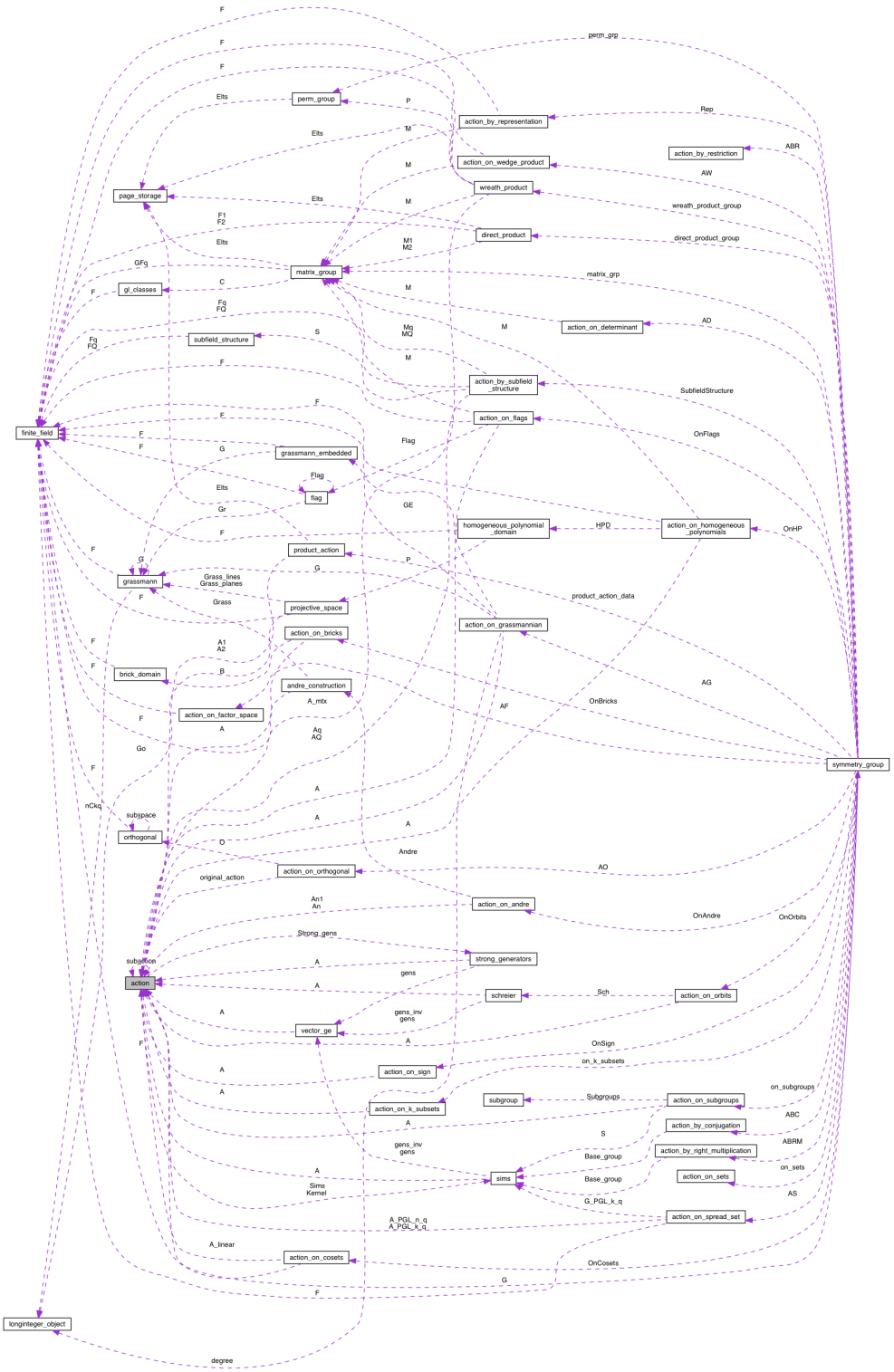

Figure 3: The collaboration graph for the action class

References

[1] Anton Betten. Classifying discrete objects with Orbiter. ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013.

[2] Anton Betten. Orbiter – a program to classify discrete objects. `https://github.com/abetten/orbiter`, 2013-2018.

[3] Cygwin. A Unix-like environment and command-line interface for Microsoft Windows. `https://www.cygwin.com`.