FROBBY USER MANUAL - VERSION 0.9.9

BJARKE HAMMERSHOLT ROUNE

Contents

1. Installation 1
1.1. Cygwin preparation 1
1.2. Mac OS 10.5 preparation using Fink 1
1.3. Linux or Mac preparation without Fink 1
1.4. Installation of Frobby 2
2. Tutorial 2
2.1. File formats 2
2.2. An example file and transformation 4
2.3. Compute the Alexander dual of an ideal 5
2.4. Hilbert-Poincaré Series 6
2.5. Intersection of ideals 7
2.6. Irreducible decomposition 8
2.7. Primary decomposition 9
2.8. Associated primes 10
2.9. Transform a polynomial 10
2.10. Maximal standard monomials 11
2.11. The Frobenius problem 11
2.12. Analyze an ideal 12

This is a minimal manual for Frobby. At present it contains information on how to install Frobby on your system, as well as documentation in the form of a tutorial which shows how to do most things that Frobby can do.

The most complete reference for the functionality of Frobby is the help system built into Frobby. However that system may be *too* complete to serve as a good introduction, and this manual is intended to cover that area.

This manual will be expanded in time to cover the algorithms that Frobby uses, explain the file formats in detail and cover the mathematical meaning of the computations that Frobby performs.

1. Installation

Frobby is tested to work on Linux, Mac OS 10.5 and Cygwin on Windows. It may work on other platforms, but that has not been tested. Although the procedure to install Frobby is the same on these platforms, Frobby requires the GMP library, which is installed in different ways depending on platform.

1.1. **Cygwin preparation.** Before installing Frobby, first run the Cygwin file setup.exe, and then install the following Cygwin packages.

libgmp-devel: In category libs. For infinite-precision integers.
make: In category devel. To build Frobby.
gcc-g++: In category devel. To compile Frobby.
diffutils: In category utils. For Frobby's test system, which uses diff.

1.2. **Mac OS 10.5 preparation using Fink.** If you have fink installed on your system, simply type

```
fink install gmp-shlibs libgmpxx-shlibs
```

1.3. **Linux or Mac preparation without Fink.** Download the newest version of GMP from `http://gmplib.org/` and unpack it somewhere. Then start a terminal, go to the directory where you unpacked it, and type

```
./configure --enable-cxx
make
make check
make install
```

Typing "make check" is optional but recommended to make sure that your system compiled GMP properly. If you have trouble installing GMP, first consult the GMP manual and then ask for help on the gmp-discuss@gmplib.org mailing list.

1.4. **Installation of Frobby.** This step is the same on all platforms. First download the latest version of Frobby from `http://www.broune.com/frobby/`. Then open a terminal, go to the directory where you downloaded Frobby, and type

```
tar -xvf frobby_v0.8.2.tar.gz
cd frobby_v0.8.2
make
make install
```

To see if your installation of Frobby is working, type

```
frobby
```

You should now be looking at information about Frobby and its help system. If you instead get a message along the lines of "command not found", type

```
bin/frobby
```

If that works, the install script was not able to place Frobby in a convenient place on your system, probably because you don't have the access rights to install programs centrally on your system. In that case you can simply call Frobby as bin/frobby from the directory where you unpacked it, or you can place the binary somewhere on your path where you do have access rights to put it.

If you want to check that your Frobby compiled correctly, type

```
make test
```

This will run a comprehensive battery of tests on Frobby, which can take a while, especially on Cygwin, which is generally a slower platform.

2. Tutorial

This is a tutorial on how to use the command line interface for Frobby.

2.1. **File formats.** All of the file formats that Frobby understands are files that will make sense to some other piece of mathematical software as well. In general Frobby will figure out by itself which of the possible formats you are using, and will produce output in that same format, so this is not something you normally have to think about. In this tutorial we will be using the Macaulay 2 format, which looks like this:

```
R = QQ[a, b, c, d];
I = monomialIdeal(
 a^2,
 a*b^5,
 c
);
```

This describes the monomial ideal $\langle a^2, a * b^5, c \rangle$ as an ideal in the polynomial ring $\mathbb{Q}[a, b, c]$. There is some syntaxtic noice in this file format, such as "I = monomialIdeal(", which is solely there so that Macaulay 2 will understand the file. In general, if the variables are $x_1, \ldots, x_n$ and the generators are $g_1, \ldots, g_k$, then the corresponding file will look like

```
R = QQ[x1,...,xn];
I = monomialIdeal(
 g1,
 ...,
 gk
);
```

It is important to note that Frobby does not understand Macaulay 2 code at all – it merely reads this very specific file format. This is why Frobby is so fussy about this precise syntax being used. E.g. this is not valid:

```
T = QQ[a, b, c, d];
I = monomialIdeal(
 a^2,
 a*b^5,
 c
);
```

The error is that it must be "R=" and not "T=", so Frobby will display the error

```
SYNTAX ERROR (format m2, line 1):
  Expected R, but got "T".
```

Likewise, this will also result in an error:

```
R = QQ[a, b, c, d];
I = monomialideal(
 a\^2,
 a*b\^5,
 c
);
```

The error is that the second i in "monomialIdeal" is lower-case, and it is supposed to be upper-case. Frobby will diplay the error

```
SYNTAX ERROR (format m2, line 2):
  Expected monomialIdeal, but got "monomialideal".
```

There is one particular error that is easy to make, namely that indeterminates in the same monomial must be separated by an asterisk *. This is because indeterminates can have names of more than one character, so otherwise there is no way to distinguish between the product of a and b and a single indeterminate named ab. Thus on this input:

```
R = QQ[a, b, c];
I = monomialIdeal(
 a^2b,
 b*c^2,
 c^3*a^3
);
```

Frobby will produce the following error:

```
SYNTAX ERROR (format m2, line 3):
  Expected ), but got "b".
```

Frobby is expecting a right parenthesis because that would make sense after the 2 on line 3, seeing as there is no * there.

Note that Frobby does not care about additional space, and it does not distinguish between a space and a newline.

To see a list of all file formats that Frobby supports, type

```
frobby help io
```

To see an example of a monomial ideal in a format X, type

```
frobby genideal -oformat X
```

If you do not specify a format, the Macaulay 2 format will be used, so if you type simply

```
frobby genideal
```

You will get something like

```
R = QQ[x1, x2, x3];
I = monomialIdeal(
 x1^3*x2^5*x3^6,
 x1^2*x2^6*x3,
 x1^5*x2^2*x3^8,
 x1^8*x2*x3^3,
 x1^5*x2^4*x3^7
);
```

However, genideal generates a random ideal, so every time you run it, you will likely get a different ideal.

2.2. **An example file and transformation.** We will need an example to run the following commands on, so take the following lines and put them into a file called "input".

```
R = QQ[a, b, c];
I = monomialIdeal(
 a^2*b,
 b*c^2,
 c^3*a^3
);
```

To check that you wrote the format correctly, try to get Frobby to read the file by typing

```
frobby transform < input
```

If you get an error from Frobby, then you didn't type the file in correctly.

Note how transform produced output in the Macaulay 2 format. If you want to transform your file into a different format, you can do so by typing

```
frobby transform < input -oformat monos
```

which will produce output in a format that can be understood by the program Monos. It will print

```
vars a, b, c;
[
 a^2*b,
 b*c^2,
 a^3*c^3
];
```

In general Frobby uses the first non-space character of an input file to figure out what format it is in, and it then produces output in that same format, unless you specify the input or output format explicitly using the command-line options -oformat and -iformat.

In general you tell frobby to do something by typing

```
frobby ACTION OPTIONS
```

where ACTION is something to do, and OPTIONS is a possibly empty list of options that specify how to do that thing. Thus you can make the transform action behave in a number of different ways depending on the options you give it. To take a look at everything that transform can do, type

```
frobby help transform
```

In general you can get information on any action by typing

```
frobby help ACTION
```

where ACTION is the name of the action. To get a list of all actions, type

```
frobby help
```

2.3. **Compute the Alexander dual of an ideal.** To compute the Alexander dual of our ideal, type

```
frobby alexdual < input
```

which produces the output

```
R = QQ[a, b, c];
I = monomialIdeal(
 b*c,
 a^2*c^2,
 a*b
);
```

And it is indeed true that the Alexander dual of $\langle a^2b, bc^2, a^3c^3\rangle$ is $\langle bc, a^2c^2, ab\rangle$.

By default Frobby will compute the Alexander dual of the input according to the point that is the least common multiple of the minimal generators of the input ideal. To use some other point, append that monomial at the end of the file.

We can append a line by using echo and cat like this:

```
echo a^10*b^10*c^10|cat input -
```

which prints

```
R = QQ[a, b, c];
I = monomialIdeal(
 b*c,
 a^2*c^2,
 a*b
);
a^10*b^10*c^10
```

so if we feed this to alexdual like this:

```
echo a^10*b^10*c^10|cat input -|frobby alexdual
```

we get the Alexander dual of the ideal according to $a^{10}b^{10}c^{10}$, which is

```
R = QQ[a, b, c];
I = monomialIdeal(
 b^10*c^8,
 a^9*c^9,
 a^8*b^10
);
```

To see the options that alexdual accepts, type

```
frobby help alexdual
```

Also note that you generally don't have to type actions out, you only have to specify a unique prefix. So to get the Alexander dual, it is sufficient to type

```
frobby al < input
```

However, it is not enough to just type

```
frobby a < input
```

Since this is ambigous with the actions assoprimes and analyze, that also have "a" as a prefix. In this case, Frobby reports that:

```
ERROR: Prefix "a" is ambigous.
Possibilities are: alexdual assoprimes analyze
```

2.4. **Hilbert-Poincaré Series.** To get the Hilbert-Poincaré series, type

```
frobby hilbert < input
```

which produces the output

```
R = QQ[a, b, c];
p =
 1 +
 -b*c^2 +
 -a^2*b +
 a^2*b*c^2 +
 -a^3*c^3 +
 a^3*b*c^3;
```

This means that the numerator of the Hilbert-Poincaré series of $\langle a^2b, bc^2, a^3c^3\rangle$ is

$$1 - bc^2 - a^2b + a^2bc^2 - a^3c^3 + a^3bc^3$$

so the series itself is

$$\frac{1 - bc^2 - a^2b + a^2bc^2 - a^3c^3 + a^3bc^3}{(1-a)(1-b)(1-c)}$$

This is the multivariate, $\mathbb{N}^n$-graded hilbert series. You may want the numerator of the univariate Hilbert-Poincaré series, which you get by typing

```
frobby hilbert < input -univariate
```

This produces the output

```
R = QQ[t];
p =
 t^7 +
 -t^6 +
 t^5 +
 -2*t^3 +
 1;
```

so the univariate Hilbert-Poincaré series is

$$\frac{t^7 - t^6 + t^5 - 2t^3 + 1}{(1-t)^3}$$

Frobby always uses the variable t for the univariate series. The univariate series is gotten from the multivariate series by substituting t for each of the variables in the ring, in this case $a = t$, $b = t$ and $c = t$.

2.5. **Intersection of ideals.** We will now intersect two ideals, so we need another file with an ideal in it. Type the following ideal into a file named input2:

```
R = QQ[d, b, c];
I = monomialIdeal(
 d*b*c,
 b^2*c
);
```

This is the ideal $\langle dbc, b^2c \rangle$. To intersect that with the ideal $\langle a^2b, bc^2, a^3c^3 \rangle$, we will need to concatenate the two files representing these two ideals. This is done by typing

```
cat input input2
```

which produces the output

```
R = QQ[a, b, c];
I = monomialIdeal(
 a^2*b,
 b*c^2,
 a^3*c^3
);
R = QQ[d, b, c];
I = monomialIdeal(
 d*b*c,
 b^2*c
);
```

If we pass this to the intersect action, we will get the intersection. So type

```
cat input input2|frobby intersect
```

to get the output

```
R = QQ[a, b, c, d];
I = monomialIdeal(
 b*c^2*d,
 b^2*c^2,
 a^2*b*c*d,
 a^2*b^2*c
);
```

Note that we now have 4 variables in the first line, since the two ideals together used these 4 variables. We see that the intersection is

$$\left\langle bc^2d, b^2c^2, a^2bcd, a^2b^2c \right\rangle$$

2.6. **Irreducible decomposition.** To get the irreducible decomposition, type

```
frobby irrdecom < input
```

which produces the output

```
R = QQ[a, b, c];
I = monomialIdeal(
 b,
 c^3
);
I = monomialIdeal(
 a^2,
 c^2
);
I = monomialIdeal(
 a^3,
 b
);
```

Note how the list of the variables does not have to be repeated because all three ideals lie in the same ring. This is saying that the irreducible decomposition of $\left\langle a^2b, bc^2, a^3c^3 \right\rangle$ is

$$\left\{ \left\langle b, c^3 \right\rangle, \left\langle a^2, c^2 \right\rangle, \left\langle a^3, b \right\rangle \right\}$$

Then it should be true that the intersection of these three ideals is the original ideal, and indeed, if we type

```
frobby irrdecom < input|frobby intersection
```

we get the output

```
R = QQ[a, b, c];
I = monomialIdeal(
 b*c^2,
 a^2*b,
 a^3*c^3
);
```

This format for a list of the irreducible components takes up a lot of space and is not the easiest to read. We can get a more compact notation by encoding each irreducible ideal as the product of its generators, noting that this is a bijection between irreducible ideals and monomials. Do this by typing

```
frobby irrdecom < input -encode
```

to get the output

```
R = QQ[a, b, c];
I = monomialIdeal(
 b*c^3,
 a^2*c^2,
 a^3*b
);
```

Here the monomial generator bc^3 corresponds to the irreducible ideal $\langle b, c^3 \rangle$.

2.7. **Primary decomposition.** Frobby can compute primary decompositions as well as irreducible decompositions. We need an example where there is a difference between these two things, so type this into a file named input3:

```
R = QQ[d, b, c];
I = monomialIdeal(
 d*b*c,
 b^2*c,
 b^10,
 d^10
);
```

To get the primary decomposition, type

```
frobby primdecom < input3
```

which produces the decomposition

```
R = QQ[d, b, c];
I = monomialIdeal(
 d^10,
 b^10,
 c
);
I = monomialIdeal(
 d*b,
 d^10,
 b^2
);
```

Note that these two ideals are primary, and that their intersection equals the original ideal (you should know how to check the intersection yourself by now).

An irreducble decomposition is also a primary decomposition, since an irreducible ideal is primary, but the irreducible decomposition can have many more elements than the primary decomposition does. In this case it has one more, since we get the irreducible decomposition by typing

```
frobby irrdecom < input3
```

which produces the output

```
R = QQ[d, b, c];
I = monomialIdeal(
 d,
 b^2
);
```

```
I = monomialIdeal(
 d^10,
 b
);
I = monomialIdeal(
 d^10,
 b^10,
 c
);
```

The primary decomposition is not unique, but Frobby computes the "nicest" primary decomposition, which is the one gotten by intersecting irreducible components of the same support, and that primary decomposition is unique.

2.8. **Associated primes.** You get the associated primes by typing

```
frobby assoprimes < input3
```

```
R = QQ[d, b, c];
I = monomialIdeal(
 d*b,
 d*b*c
);
```

Each generator encodes an associated prime. In this case the associated primes are $\langle d, b\rangle$ and $\langle d, b, c\rangle$. You can check that these are the radicals of the primary components,

2.9. **Transform a polynomial.** Transform does a number of things to ideals, and ptransform is the corresponding action for polynomials. Put a polynomial into the file pinput by typing

```
frobby hilbert < input > pinput
```

Then type

```
cat pinput
```

to get

```
R = QQ[a, b, c];
p =
 1 +
 -b*c^2 +
 -a^2*b +
 a^2*b*c^2 +
 -a^3*c^3 +
 a^3*b*c^3;
```

This is a file that Macaulay 2 can understand. To change the format of this file to something that Singular can understand, type

```
frobby ptransform < pinput -oformat singular
```

to get

```
ring R = 0, (a, b, c), lp;
int noVars = 0;
poly p =
 1
```

```
 -b*c^2
 -a^2*b
 +a^2*b*c^2
 -a^3*c^3
 +a^3*b*c^3;
```

You might notice that the Singular format has something called noVars. This is there merely because all Frobby formats must be able to represent a ring with no variables, and Singular has no such concept. Thus Frobby gets around that by always giving the ring at least one variable, but writing noVars=1 if there really should not be any, and noVars=0 otherwise.

To get the polynomial into a format that 4ti2 might have produced, if it had a concept of polynomials, type

```
frobby ptransform < pinput -oformat 4ti2
```

to get

```
6 4
1  0 0 0
-1  0 1 2
-1  2 1 0
1  2 1 2
-1  3 0 3
1  3 1 3
(coefficient) a b c
```

The first line indicates the size of the matrix, and each row of the matrix is a term, with the first number being the coefficient, and the remaining numbers being the exponent vector. I.e. the exponent of a, b and c respectively in that order.

To see what else ptransform can do, type

```
frobby help ptransform
```

Note that if we knew in advance that we wanted to change the format to, say, 4ti2 format, we could have produced the file pinput in that format directly by typing

```
frobby hilbert < input > pinput -oformat 4ti2
```

2.10. **Maximal standard monomials.** To produce the maximal standard monomials, type

```
frobby maxstandard < input3
```

to get

```
R = QQ[d, b, c];
I = monomialIdeal(
 d^9*b^9
);
```

2.11. **The Frobenius problem.** Given an input vector $p = (p_1, \ldots, p_n)$ of positive relatively prime integers, the Frobenius number of p is the largest integer that cannot be written as a linear combination of $p_1, \ldots, p_n$ where the coefficients in the combination are non-negative integers.

To work with this, put the following example in a file named frob

```
6 10 15
```

and type

```
frobby frobdyn < frob
```

to get as output that the Frobenius number is

```
29
```

The frobdyn action uses an obscenely slow dynamic programming algorithm to compute the Frobenius number. If you install the program 4ti2 and put it into the empty 4ti2 subfolder of the Frobby folder, you can use an algorithm that is much faster when the Frobenius number itself is moderately large.

Assume that you have installed 4ti2 in that folder. Then, within the Frobby folder, type

```
./frobgrob frob
```

which will produce the same output, namely

```
29
```

Now put this input into a file named frob2

```
1234567890001
348461546433
6484646532513541
45464188888115164
156148465156186446846531O
```

and type

```
./frobgrob frob2
```

to get that the Frobenius number is

```
15111053020091472900
```

This computation would never have completed using the dynfrob action. Note that you get better performance if you sort the numbers in increasing order.

Note that ./frobgrob is a script that uses 4ti2 and the frobgrob action of Frobby. This is a bit annoying to get right, which is why there is a script doing it. To get a hint on how to do this yourself directly without using the script, take a look at the script, look at the documentation for 4ti2, and type

```
frobby help frobgrob
```

to get an idea about what input frobgrob expects.

You can also generate random Frobenius instances by typing

```
frobby genfrob
```

to produce something like

```
4259 8346 12295 18936 22645 27086 34182 69298 74617 84570
```

The output is random, so you will likely get a different instance.

2.12. **Analyze an ideal.** To get Frobby to analyze the ideal and tell you what it figured out about it, type

```
frobby analyze < input
```

which produces the output

```
3 generators
3 variables
```

This is admittedly not an impressive level of analysis being performed. To see a few more things that analyze can do, type

```
frobby help analyze
```

The idea is that this action will be expanded to provide actual valuable information, such as whether the ideal is strongly generic, if it is weakly generic, strongly/weakly co-generic, how many irreducible and primary components it has, if any of the given generators are non-minimal and various things like that.