Random Comments on polymake's Design

Michael Joswig

June 21, 2017

This is intentionally kept short rather than complete. All examples refer to version 3.1 of March 2017.

1 Types and defined semantics for all objects

polymake is software for the working mathematician. Therefore, its goal is to provide proper descriptions of valid mathematical concepts. The user experience should be as seemless as possible. In particular, calling a function on an object should be independent of how the object was initially defined.

The first example is a polytope, in fact, just a quadrangle in the plane. Note that polymake employs homogeneous coordinates (thus the leading ones).

```
In [1]: $P = new Polytope(VERTICES=>[[1,1,0],[1,0,1],[1,3,2],[1,5,7]]);
```

Any computation might trigger calling third party software (lrs in the example below) on the way. polymake takes care of translating from its own data description into the description of the third party software and back. While this is no big deal in the example, we keep this paradigm throughout.

```
In [2]: print $P->F_VECTOR;
```

```
Out[2]: polymake: used package lrs
          Implementation of the reverse search algorithm of Avis and Fukuda.
          Copyright by David Avis.
          http://cgm.cs.mcgill.ca/~avis/lrs.html

        4 4
```

polymake is organized into 'applications' which essentially serve as namespaces. The application 'fulton' deals with toric varieties.

```
In [3]: application "fulton";
```

The seemless integration allows to carry objects over application boundaries. Here we produce a toric variety.

```
In [4]: $T = new NormalToricVariety($P);
        print $T->PROJECTIVE;
```

```
Out[4]: polymake: used package ppl
          The Parma Polyhedra Library (PPL): A C++ library for convex polyhedra
          and other numerical abstractions.
          http://www.cs.unipr.it/ppl/

        polymake: used package cdd
          cddlib
          Implementation of the double description method of Motzkin et al.
          Copyright by Komei Fukuda.
          http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html

        1
```

Types of objects and their subobjects can be read off. `polymake` establishes a hierarchy among its types, including multiple inheritence and more.

```
In [5]: print $P->TRIANGULATION->type->full_name();

Out[5]: GeometricSimplicialComplex<Rational> as Polytope<Rational>::TRIANGULATION
```

Note that the polytope P belongs to the application 'polytope', while we currently sit in 'fulton'. The property TRIANGULATION works like a data member of the polytope object P. It is a SimplicialComplex, which is an object type in yet another application called 'topaz'. Hence we can compute the (reduced integer) homology.

```
In [6]: print $P->TRIANGULATION->HOMOLOGY;

Out[6]: ({} 0)
        ({} 0)
        ({} 0)
```

`polymake`'s object model is explained in

> Gawrilow and Joswig: Flexible object hierarchies in `polymake`, Mathematical software — ICMS 2006, Lecture Notes in Comput. Sci. 4151, pp. 219-221 (2006), http://dx.doi.org/10.1007/11832225_20.

2 Perl, yet enhanced

There are many programming languages, and most of them have at least some useful features which distinguish them from others. From the very beginning `polymake` was designed as a hybrid employing C++ (as a compiled language) and Perl (as an interpreted language). This combines interactivity with fast execution. Systems which follow similar ideas today often rely on some form of just-in-time compilation. My current favorite among the implementations that I am aware of is the one in Julia.

`polymake`'s approach, which can be superior to just-in-time compilation if done right, is based on our extension of the Perl language by a number of additional features. Everything is based on the idea that calling a function (which looks like Perl) in `polymake` interpreter might actually be implemented in C++. This way, e.g., long integer arithmetic (implemented in GMP) becomes available in Perl, including overloading of operators.

```
In [7]: print fac(120)+fac(5);

Out[7]: 6689502913449127057588118054090372586752746333138029810295671352301633557244962989366874
```

The most important modification is is bringing C++ templates to Perl. These are type parameters. However, in modern dialects of C++ the sub-language for the templates is Turing complete in itself.

As an example we look at ordered sets of (machine size) integers.

```
In [8]: $S = new Set<Int>();
        for (my $i=0; $i<10; ++$i) { $S += new Int(rand()*100) }
        print $S;

Out[8]: {18 40 48 58 66 70 77 87 93 96}
```

The point is that the same C++ code which takes care of the sets of integers also deals with other types, such as sets of vectors of rational numbers. The default ordering is lexicographic. We switch back to the application 'polytope'.

```
In [9]: application "polytope";
        $C = cube(5);
        $S = new Set<Vector<Rational>>();
        for (my $i=0; $i<10; ++$i) { $S += $C->VERTICES->[new Int(rand()*32)] }
        print $S;

Out[9]: {<1 -1 -1 1 -1 1> <1 -1 1 1 -1 1> <1 -1 1 1 1 1> <1 1 -1 -1 -1 1> <1 1 -1 1 1 1> <1 1 1
```

It is by no means obvious what the advantage over just-in-time compilation should be here. The difference comes from how the Set class is implemented in `polymake`'s C++ template library: our C++ code avoids using virtual functions and virtual classes. This is beneficial in terms of speed on modern hardware due to better caching behavior.

Most of `polymake`'s core algorithms are templated. This includes convex hulls computations and linear optimization. Below is an example computation concerning a polytope defined over a field of formal Puiseux series with real coefficients.

```
In [10]: set_var_names<UniPolynomial<Rational,Rational>>(qw(t));
         $monomial = monomials<Rational,Rational>(1);
         $t = new PuiseuxFraction<Min>($monomial);
         print klee_minty_cube(3,$t)->VOLUME;

Out[10]: polymake: used package tosimplex
           Dual simplex algorithm implemented by Thomas Opfer

         (1 -2*t + t^2)
```

The volume is a rational function in the parameter t (in fact, here it is a polynomial). This is also valid for real polytopes parameterized by t for sufficiently small positive values of t.

3 XML based file format, standardized via RELAX-NG

Since computations can be costly, storing the data, including intermediate results, may be vitally important.

```
In [11]: $R = rand_sphere(5,300);
         prefer_now("beneath_beyond"); $R->FACETS;
         print $R->F_VECTOR;
         save $R, "R.poly";

Out[11]: 300 4468 14892 17870 7148
```

The following does not require any computation.

```
In [12]: $R = load "R.poly";
         print $R->F_VECTOR;

Out[12]: 300 4468 14892 17870 7148
```

As a recently introduced feature `polymake` can provide an entirely self-contained description of its own file format as RELAX-NG. This allows for interpreting `polymake` data independent of the `polymake` software. This also offers the possibility to write generic interfaces to `polymake`. This might be particularly useful since the precise file format is constantly changing (to be able to keep up with any progress in the mathematics underlying the system).

```
In [13]: save_schema $R, "R.rng";

In [14]: system "head -30 R.rng";

Out[14]: <?xml version="1.0" encoding="utf-8"?>

         <grammar datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes" ns="http://www.ma
           <include href="file:///data/polymake/git/xml/common_grammar.rng" />
           <define name="PolymakeVersion">
             <attribute name="version">
               <value>3.1</value>
             </attribute>
           </define>
           <start>
             <ref name="polytope-Polytope.Rational" />
           </start>
           <define name="polytope-Polytope.Rational">
             <element name="object">
               <ref name="PolymakeVersion" />
               <attribute name="type">
                 <value>polytope::Polytope&lt;Rational&gt;</value>
               </attribute>
               <ref name="Extensions" />
               <ref name="polytope-Polytope.Rational-contents" />
```

```
  </element>
</define>
<define name="polytope-Polytope.Rational-contents">
  <ref name="ObjectTextDescriptions" />
  <oneOrMore>
    <element name="property">
      <choice>
        <ref name="polytope-Polytope.Rational.LATTICE_BASIS" />
        <ref name="polytope-Polytope.Rational.SMOOTH" />
        <ref name="polytope-Cone.Rational.MONOID_GRADING" />
```

Gawrilow, Hampe and Joswig: The `polymake` XML file forma}, Mathematical software — ICMS 2016. 5th international congress, Berlin, Germany, July 11–14, 2016, pp. 403-410, http://dx.doi.org/10.1007/978-3-319-42432-3_50.

Stay tuned and: Happy polymaking!