PALP++ project proposal

by Maximilian Kreuzer, Vienna, September 2010

Abstract

PALP [http://arxiv.org/abs/math/0204356] is A Package for Analyzing Lattice Polytopes with Applications to Toric Geometry. While technically concerned with discrete and algebraic geometry, its development has mostly been driven by applications to string theory. Here I'ld like to present some ideas about how a dynamical and object oriented version of PALP, called palp++, might look like and how it might be designed and implemented by a team of users and contributors so that palp++ would grow along with its applications like PALP did for many years, but overcoming the design limitations that make further PALP development increasingly painful. The basic idea (motivated by Ilarion Melnikov's suggestion at the AEI conference on (0,2) Mirror Symmetry 2009 to organize a PALP developer's workshop) is to have a team discuss and define the desirable specifications, interfaces and data structures before writing any code. Then, after implementing a very basic core, the various tasks could be implemented by contributors motivated by their needs while applying the code for their respective research work (migrating code from PALP and/or rewriting, improving or extending the various routines). In the process a core team, deciding on standards and releases, should naturally emerge. I think a discussion wiki would be enough for the start and a workshop could be more efficient at some later point in time. palp++ (like PALP) should be put under the GNU licence.

1 Some general considerations

I hope that a realistic approach will be to start out with a discussion wiki located, for example, at the address `palp.itp.tuwien.ac.at/wiki`, and the development environment `git`, which Anton.Rebhan@tuwien.ac.at both volunteered to set up. The first thing of palp++ that hence should exist is a basic documentation, where objects and commands could have a status like: projected / testing / released.[1]

I know too little about C++ and leave details to be discussed by others, but what I would like to insist on is to have `palp++` written in C++ and be independent of libraries except for the presumably unavoidable `gmp++` and `blitz++`, while doing, e.g., commutative algebra (CA) for intersection rings by interacting with Singular (via program calls and communication via pipes) in such a way that only a working Singular installation is required, so that an error message/exit occurs only for routines that need CA while other users/applications are never concerned with worrying about a Singular installation. palp++ should thus be as safe and easy to compile as possible, like PALP, with executables also working well in pipelines and batch jobs in order to perform tasks like finding polytopes with a number of special properties in large lists.

PALP++ should work with integers, rather than with floating, so that it (in principle) is able to produce "exact" results. What makes PALP fast (in addition to the use of binary operations on bitwise encoded INCIdences for avoiding computationally expensive linear algebra) is its use of 32 or 64 bit arithmetics rather than arbitrary precision. This makes it vulnerable to overflows. But often there exist much faster algorithms for checking the correctness of a result then for computing a (candidate) solution. When working on large lists a good strategy could thus be to compute generically with low precision and redo the calculation with higher or arbitrary precision only when necessary. I hope that things like "templates" in C++ make this feasible. In the best of all worlds palp++ thus would have options "sloppy" vs. "exact=strict" with "strict" versions of the algorithms becoming available for a (with time increasing) subset of the routines. These would first try with 32 bit or estimate what is needed and if this fails recompute or finish the calculation with the necessary precision (like 64 bit, possibly 128 bit when supported by modern processors or with tricks like computing modulo large primes, and only if necessary the computationally expensive `gmp++`.

While I did have bad experiences with gmp++ libraries, like producing wrong results[2], gmp++ probably is unavoidable and I hope that the problem of inadvertedly getting wrong results because of library incompatibilities has by now become history or that people know how to avoid troubles. By the way: Options/flags should not only be available at program call but also (like basic assignment of objects with names) on input lines, so

[1] Analogously, a web address `cy.itp.tuwien.ac.at/wiki` could complement/include (an updated version of) the current data page `http://hep.itp.tuwien.ac.at/~kreuzer/CY/` in order to enable user input.

[2] I hence included the simple check

```
{ ULong N[6]={7,11,3,4,5,9}; cout << MultiNomial(30,N,6) << endl;}
cout << "161693309933255413272960000 == 39!/(7!*11!*3!*4!*5!*9!)";
to tell me whether I could trust the executable in my first C++ program which I wrote for finding PF operators.
```

that work can be done on single polytopes without restarting everything again and again. But only to a very basic extent leaving the writing of palp++ interfaces for higher level languages like Singular, the SAGE project or the commercial Mathematica or Maple, packaging for *deb etc. as very welcome activities but independent of the core palp++ project. Notwithstanding, of course, contributions to the wiki discussions and program definition wish lists for facilitating such activities are very desirable.

Like the linking of gmp++, the use of some matrix multiplication library may be recommendable. If we want to work over integers, I got the impression that blitz++ is the unique choice. I leave it to the experts to discuss this.

2 IO, objects and executables

PALP++ should work with cones, i.e. with the ray representation of polytopes. Cones C can be pointed, maximal dimensional, Gorenstein (i.e. generated at degree 1 and hence equivalent their generating integral polytopes P_C) or reflexive Gorenstein of index r (in which case $r\,P_C$ is reflexive up to a shift). Accordingly, facet/vertex enumeration may have different skopes. For the analysis of integer polytopes in their ray representation with known grading PALP's FindEquations can directly be generalized, but for more general cones (like Mori cones) some preprocessing needs to be done (see below).

There could, for example, be 3 executables called "cone" (or "cone.x"), "cicy" and "enum[erate]", with the latter replacing class.x and cws.x. (By the way: I propose to introduce the synonyms "weight vector" for "weight system" and "weight matrix" for "combined weight system" as I did in arXiv:0809.1188 [math.AG]; introduciing both sets of names should avoid any confusion). "cone" would essentially be a subset of the other two, where it is a matter of taste whether to put toric hypersurfaces also into cone (like for poly.x now) or having cone only do combinatorics and put all toric geometry applications into cicy. One may also prefer to have everything in a single executable.

In the best of all worlds input would be compatible with PALP, but at the same time with important systems like POLYMAKE and/or cdd input formats. This is probably not consistent.[3] In any case weight matrix input should work for polytopes/cones. In the long run it would be very convenient to have multidegree input for complete intersections, and also some other kinds of objects should be readable, like for avoiding unnecessary recomputations in pipelines, or specifying a particular triangulation or Mori cone for which further data should be computed. To enable this, these kind of data could be distinguished by keywords. We will also need "comment lines" for printing other information that is not supposed to be readable. Mori cones are examples of cones that are not Gorenstein, so we will need a version of FindEquations (i.e. facet enumeration) that works on more general cones, possibly not pointed and not maximal-dimensional. How far one can go with defining toric data via multigradings, Gale transforms, Mori data, line bundle data for non-CY's, etc. will require some hopefully interesting math discussions.

While I think all these things should be discusssed in detail before deciding on the specifications, for the implementation of a first core program it will be good enough to have facet and lattice point enumeration for Gorenstein cones (and polytopes) with some preliminary IO and to have everything else implentented around it later on. A constant pain in the neck in PALP is the question of whether polytopes live "naturally" in the M or in the N lattice. The choice of the M-lattice is related to the logic of the weight matrix input, while the N-lattice is often the more relevant polytope in toric geometry applications (in case of known weight matrices it will be desirable to keep the info how the WM-columns are related to rays in the N-lattice). In palp++ this should be solved automatically by having all kinds of data being members of a big object, whose sub-objects are void until they are computed and stored as they are needed during the required calculations. Such objects could be "Gcone" (with index=0 for the non-reflexive case), "Cone" for general cones, etc.; this suggests to have the "original" cone associated to the M-lattice (or rather $\bar{M} = M \oplus \mathbb{Z}$) and its dual to N (nevertheless the data of C can remain void during the whole calculation if, e.g., the input is declared to be the dual cone).

[3] PALP's convention that coordinate matrices can be line or column, assuming that the points span the dimension, prohibits, e.g., the computaion of the baricenter and relative volume of faces with `poly.x -B` (help on options that were implemented after the original publication is available with the option `-x` for e**x**tended or e**x**perimental).
Still, column vector IO is very useful as long as it fits the line, so one chould keep the choice like with keywords CV vs. LV, or even add CR and VR for ray representations with columns vs. lines.

3 Some details on algorithms and routines

Two basic routines of PALP are `Find_Equations()` and `Complete_Poly()`. They can be found in *Vertex.c*, which also contains the INCIdence calculus with fast bitwise operations that help avoid expensive linear algebra.

- `Find_Equations()` computes the bounding equations $\vec{a}\vec{x}+c\geq 0$ of a polytope P, i.e. the vertices $\vec{a}/c$ of the dual polytope P^* with $\langle P,P^*\rangle\geq -1$. In the ray representation this corresponds to duality of cones $\langle C_P, C_P^\vee\rangle\geq 0$ with $(\vec{x},1)\in C_P$ and $(\vec{a},c)\in C_P^\vee$. In an arbitrary basis there is no more distinction between points and equations; they are all (generators of) rays.

* `Double description algorithm`: The algorithm used by PALP is of the "double description" type, which approximatex the polytope from inside. It first finds a startsimplex consisting of vertices of P that are found be extremizing coordinates. This initializes a list of vertices and of candidate facets/equations. Each "candidate equation" is then either a facet of P or determines a new vertex that is found by extremization of coordinates among the points violating $\vec{a}\vec{x}+c\geq 0$. In the latter case the convex hull is enlarged by a new vertex and the "candidate equation" is replaced by a number of new "candidate equations". The procedure terminates when no more "candidate equations" are left.[4]

In the two years between our enumeration of 4d reflexive polytopes and the first publication of PALP we studied numerical stability and high-dimensional behavior in order to make PALP safer for applications like CY 4-folds:

- Numerical stability and `W_to_GLZ();` in *Rat.c*:
  I came to the conclusion that all numerical instabilities that showed up in examples are essentially related to the iterative computaion of the extended greatest common divisor (EGCD), i.e. of a vector $\vec{A}$ that obeys $\vec{A}\vec{W}=g$ where g is the greatest commod divisor of the components of $\vec{W}$. The solution I came up with is to improve the coefficients $\vec{A}$ in each step of the iteration by addding solutions of the homogeneous equuation. Closer inspection shows that this is equivalent to the determination of a GLZ-matrix (i.e. a lattice automorphism) G that transforms $\vec{W}$ into g times the first basis vector with G approximately triangular with minimal off-diagonal entries. $\vec{A}$ is the first line of G and its other lines are determined in its iterative computations. For details inspect the code of `W_to_GLZ()` in the file *Rat.c*

  In my contribution to the code of PALP I eventually did all computations over integers and all changes of basis are composed of GLZ-transformations determined in the above way. In particular, the name of the auxiliar routine `GLZ_Start_Simplex()` indicates the re-implementation of the linear algebra needed for finding a start-simplex in order to stabilize the numerics.
- The only computation in `Find_Equations()` that could not be stabilized with this idea was the routine `EEV_To_Equation` that computes a new candidate equation in terms of a new vertex and two candidate equations that intersect in a codimension-2 face. Here intermediate expressions can be of the order of the square of input and output entries and hence should be computed with double precission.
- `Pivoting` in the DD algorithm refers to the selection of the candidate equation that is used for the determination of the next vertex (and the selection of that vertex). It becomes essential for higher-dimensional polytopes. While a number of parameters in the code that are set to 0 are remnants of experiments that did not make things better the simple pivoting strategy to the selected the next CEq among of the newly generated ones (i.e. keep on working where you are) turned out to be good enough to become competative with *cdd* in examples of dimension 15 or so.[5] The setting of the flag `MAX_BAD_EQ` shows that this pivoting should be switched on above dimension ~ 6.

* `PALP++ version`: Vertex The above equations amount to a grading vector $(\vec{0},1)$. Generalization

FindFacets: assert rays!=0; add 0 and find startsimplex; make 0 vertex; [0 <= $ax+c$:: ignore $c>0$, $c<0$ bad, $c=0$ cand-facet; $0\in conv(rays)$ => not strongly convex !! =¿ projection ...] determine equations and span-basis; find facets; Startsimplex should always find 0=vertex if 1st coordinate is positive. ... should apex be 0th ray in ray.pointer.list?

[4] IPcheck and Refcheck versions terminate already when a facet equation with $c=\leq 0$ or $c\neq 1$ is found. For PALP++ these routines are presumably not needed, except possibly for an IPcheck in the context of enumeration problems.

[5] Svozil, correlation polytopes for Bell'type inequalities

  - Should identical rays and 0 be removed or forbidden? Non-primitive be reduced? I.e., should we store the input "as is" or only "useful" information?
  - IPCheck and RefCheck versions?

- `Incidences`:

- `Complete_Poly` was written by Harald and seems to be very efficient lattice point completion. Can this be used instead of Newton polytopes (NP) for weight matrix input (in the input routine Read...)? Or is there a way to get generators of C and $C^\vee$ from WM input without any NP point completion so that lattice point completion can be avoided on both sides also for WM input unless needed for other purposes?

- Ordering of points? Vertices first? Points according to codimension / interior to faces ...; position of interior point choice of output options

WARNING: identical points in input (removed) / 0 removed; ERROR: 0 in interior EnumVF.c Sortadd.c Compress.c NormalForm.c Binlist DBlist

From a mail by Volker Braun: pipe() generate read/write pipes erzeugen, with fork()/exec() subprozess, then in child-prozess with dup/dup2 connect pipes with subprozesse I/O.

[6]

4 Various details

- INCIs: reverse order! (cf. Mori.c), count 0/1; positions of vertices

- Triangulations:
  Projective ambient space / triangulate facets; phases on CY: 2-skeleton
  vs. Andre's counterexample.
  An important task will be the lifting of the triangulation of the ambient space of the base to the 4-fold and study of possible singularities.

- CICY classification via reflexive Gorenstein cones

objects: cone [ray face inci] template :: precision/bits Index=-1=none, 0=Gorenstein, 1=Reflexive, ¿1=Ref.Gorenstein DualGorenstein ??

tasks cf. Global.h and docu of with -h for help on things that existing in the first release 2002 and -x for extensions/experimental for stuff I added later. Mori stuff is still inofficial but its status and code is known by the Vienna F-theory band; staring with base of elliptic 4-fold with fans/ambient spaces restrected by reflexive polytopes, with all new code contained in a file called Mori.c.

cone -¿ points npoints vertices ndualpoints dualvertices inci [star] triang enum -¿ polylists (RAMLIST / BINLIST / DBLIST) WS CWS ... cicy -¿ & nefpart hodge mori

DOCU: tasks and commandlist status: projected (in PALP?) / beta / testing / released

Programs: cone -¿ points npoints vertices ndualpoints dualvertices inci [star] triang enum -¿ polylists (RAMLIST / BINLIST / DBLIST) WS CWS ... cicy -¿ & nefpart hodge mori

5 Mailing list

Please suggest people I may have forgotten:
`rebhana@hep.itp.tuwien.ac.at`
`skarke@hep.itp.tuwien.ac.at`
`knapp@hep.itp.tuwien.ac.at, johanna.knapp@ipmu.jp`
`walliser@hep.itp.tuwien.ac.at`
`cmayrh@hep.itp.tuwien.ac.at`

[6] High-dimensional G-cones with large symmetries [aks Karl Svozil for examples related to Bell-inequalities in Quantum Mechanics] FindVFS(): list vertices (all), facets (orbits) and symmetries (generators) find: start facet (large/small?), incident facets (decide: new or automorphic) question: find subgroups and extensions, efficient representation of VertPerm make: list of incident facets, identify translates via incident cd2-NF

abraun@hep.itp.tuwien.ac.at
Ilarion Melnikov <ilarion.melnikov@gmail.com>
Emanuel Scheidegger <emanuel.scheidegger@math.uni-augsburg.de>
Benjamin Nill <bnill@math.uga.edu>
a.m.kasprzyk@usyd.edu.au
batyrev@mail.mathematik.uni-tuebingen.de
Duco van Straten <straten@mathematik.uni-mainz.de>
Gert Almkvist <gert.almkvist@yahoo.se>
Lev Borisov <borisov@math.rutgers.edu>
Vasily Golyshev <golyshev@mccme.ru>
lukas@physics.ox.ac.uk
seung.lee@chch.ox.ac.uk
jamesgrayphysics@gmail.com
yang-hui.he@merton.ox.ac.uk
andlara@hep.physics.upenn.edu
Philip Candelas <pcandelas@mac.com>
Xenia de la Ossa <delaossa@maths.ox.ac.uk>
Charles Doran <doran@math.ualberta.ca>
Andrey Novoseltsev <novoselt@gmail.com>
Timothy G Abbott <tabbott@MIT.EDU>

Appendix

A.1 Symmetric polytopes and dimensional descent

A.2 Conifold CYs and enumeration of topologies

References

[1] M. Kreuzer and H. Skarke, "PALP: A Package for analyzing lattice polytopes with applications to toric geometry," Comput. Phys. Commun. **157** (2004) 87 [arXiv:math/0204356].

[2] M. Kreuzer, "Toric Geometry and Calabi-Yau Compactifications," Ukr. J. Phys. **55** (2010) 613 [arXiv:hep-th/0612307].

[3] V. Batyrev and B. Nill, "Combinatorial aspects of mirror symmetry," arXiv:math/0703456.

[4] B. Nill and J. Schepers, "Gorenstein polytopes and their stringy E-functions," arXiv:1005.5158 [math.CO]

[5] M. Kreuzer, "On the Statistics of Lattice Polytopes," arXiv:0809.1188 [math.AG].

[6] C. M. Chen, J. Knapp, M. Kreuzer and C. Mayrhofer, "Global SO(10) F-theory GUTs," arXiv:1005.5735 [hep-th].

[7] A. Collinucci, M. Kreuzer, C. Mayrhofer and N. O. Walliser, "Four-modulus 'Swiss Cheese' chiral models," JHEP **0907** (2009) 074 [arXiv:0811.4599 [hep-th]].

[8] V. Batyrev and M. Kreuzer, "Constructing new Calabi-Yau 3-folds and their mirrors via conifold transitions," arXiv:0802.3376 [math.AG].

[9] A. Klemm, M. Kreuzer, E. Riegler and E. Scheidegger, "Topological string amplitudes, complete intersection Calabi-Yau spaces and threshold corrections," JHEP **0505** (2005) 023 [arXiv:hep-th/0410018].

[10] V. Braun, M. Kreuzer, B. A. Ovrut and E. Scheidegger, "Worldsheet Instantons and Torsion Curves, Part B: Mirror Symmetry," JHEP **0710** (2007) 023 [arXiv:0704.0449 [hep-th]].