CoCoALib and CoCoA-5

J. Abbott, Universität Kassel
A. M. Bigatti, Università degli Studi di Genova

abbott@dima.unige.it
bigatti@dima.unige.it


Introduction

The CoCoA project dates back to 1987, with the first public release of its interactive system in 1989. The aim has always been to provide a user-friendly software laboratory for studying Computational Commutative Algebra, specifically ideals of multivariate polynomials (*e.g.* Gröbner bases).

Starting in the early 2000s the CoCoA software has undergone a profound change in its internal design: its "mathematical expertise" resides in *CoCoALib*, a C++ software library [5]; there is also an interactive system *CoCoA-5* [9] which uses an interpreter to grant easy access to CoCoALib's capabilities. All code is free and open source (licence: GPL-3).

We give an overview of the latest developments in the library and system: specifically for the recent release of CoCoA-5.2.0/CoCoALib-0.99550 (May 2017), and the upcoming release (autumn 2017).

The maturity of the software means that the overall design of CoCoALib and CoCoA-5 is now quite stable, so the main changes are the addition of numerous new functions. There are, nevertheless, a few handy changes in the user experience: *e.g.* verbose mode, interrupt handling, a time-out mechanism, and a procedure for "graceful obsolescence".

Verbose mode

There are several circumstances where it can be useful to know what is happening inside a function call: *e.g.* to understand how the computation is progressing (and whether it will likely finish in a reasonable time), for developers to check that intermediate results are correct, and even for educational purposes.

The new function `SetVerbosityLevel(N)` sets the global verbosity level to `N`. Various functions defined in CoCoALib and in the CoCoA-5 packages will print out some internal progress messages when the global verbosity level is higher than a certain threshold value: higher levels trigger greater verbosity. For instance, the lowest level giving information on the progress of Gröbner bases is 100: every time a new basis polynomial is found, an informative line is printed:

```
/**/ SetVerbosityLevel(100);
/**/ GBasis(ideal(x^4-z^2, x^3-y));
myDoGBasis: New poly: len(GB) = 1  len(pairs) = 1
myDoGBasis: New poly: len(GB) = 2  len(pairs) = 1
...
myDoGBasis: New poly: len(GB) = 5  len(pairs) = 2
```

Besides `GBasis`, verbose information can be produced by numerous other functions, and can also be activated within user defined functions.

Since Gröbner basis computation is quite pervasive in CoCoA its verbosity threshold level is quite high, whereas in general other functions have levels in the range 10–99. The values 1-9 are effectively reserved for user defined functions, so that they may be used without triggering any verbosity from CoCoA internal functions.

The number of functions which respond to the verbosity setting is steadily increasing — details are in the documentation (*e.g.* in CoCoA-5 type "`?verbose`").

Interrupt handling

In some hard Gröbner basis computations, by setting the verbosity level, one may see that the number of pairs yet to be processed is getting unfeasibly high. The user may then choose to interrupt the computation by typing `Ctrl-C`: the computation will be interrupted as soon as the reduction of the current S-polynomial terminates.

This interruption cancels the incomplete Gröbner basis computation, and returns the computer to the state it was in just before the Gröbner basis computation was begun. This allows the user to continue using the CoCoA-5 session safely, *e.g.* using values computed and stored before the interrupted function call. In other words: there is no need to fall back on the popular technique "turn it off and on again" ;-)

Implementing this correctly is not trivial: thanks to its clean, exception-safe design, CoCoALib guarantees that no memory has been lost, and no data has been modified.

All code written in the CoCoA-5 language is intrinsically interruptible thanks to the design of the interpreter. Those who write C++ and use CoCoALib directly have a similarly comfortable situation. The only point to note when writing C++ code is that a normal function in C++ "ignores" interrupts, so to make it interruptible there has to be an explicit check for a pending interrupt signal. The design for inserting such a check in potentially critical loops in C++ code using CoCoALib is quite simple: it is enough to add a line like this

```
CheckForInterrupt("myFunctionName");
```

this function checks if the "interrupt flag" has been set, and if so, throws an exception. Thus it may be used in any function in CoCoALib (even those defined by users of CoCoALib).

A small aside: one might like to interrupt a computation, perform some checks and then decide whether to continue the computation from that stage. This option requires keeping in memory a (possibly huge) partial computation for further access, and is not (yet) implemented.

Time-Outs

Gröbner basis computations are notorious for being unpredictably lengthy: two seemingly similar inputs can lead to wildly varying computation times. A new feature in CoCoALib (and thus also in CoCoA-5) is the ability to set a *time-out*, an upper limit for the amount of time allowed to complete a computation: if it takes longer than the specified limit then a *time-out error* is generated.

The time-out mechanism is already available in an interim release, and will be in the next stable release CoCoA-5.2.2/CoCoALib-0.99560. The design allows time-outs to be applied easily to almost any C++ function using CoCoALib.

One application of time-limited Gröbner basis computations is in SMT solving with polynomial equalities. In this context, speed of computation is important; if a Gröbner basis can be obtained *quickly* then it can be used to assist future decisions and computations (or perhaps to prove unsatisfiability).

To illustrate the ease of using the time-out feature, here is the CoCoALib code for a time-limited Gröbner basis computation (based on the time-unlimited function `GBasis`). This function will either return the Gröbner basis or throw a `TimeoutException` if the computation is taking longer than `MaxTime` seconds.

```
auto GBasisTimeout(const ideal& I, double MaxTime)
{
  CpuTimeLimit timeout(MaxTime);
  return GBasis(I);
}
```

Graceful Obsolescence

As software develops, sometimes a few old functions become obsolete because a better design has evolved: *e.g.* perhaps a new more general function, or a more expressive syntax. For some users this "progress" is a deterrent to updating to the latest software release, for fear that their programs would stop working.

In CoCoA-5 and CoCoALib we have opted to first declare such functions *obsolescent*, *i.e.* they will become *obsolete* in a few years time. Such functions continue to work, but produce helpful warning messages about how the call should be updated. Here is an example in CoCoA-5. Imagine that a user runs these lines from an old file

```
/**/ Weights := matrix([[1,2,3,4]]);
/**/ M := CompleteToOrd(Weights);
```

The variable `M` will be assigned the expected result, but a warning message is also printed:

```
--> WARNING: "CompleteToOrd" is obsolescent;
--> WARNING: use "MakeTermOrd" instead.
```

We use this approach to allow users time to update their programs, and to provide useful guidance on how to do this. The actual implementation of the obsolescent functions resides in the special files: `obsolescent.cpkg5` for CoCoA-5 functions, and `obsolescent.[HC]` for CoCoALib functions.

News specific for CoCoALib

For simplicity we describe the new CoCoALib functions below in the following section on CoCoA-5, since most of the new functions added to CoCoA-5 are actually part of CoCoALib.

Special effort has always been invested in making CoCoALib clean and portable. One new feature specific to CoCoALib is the improved, more portable *configure-build-install* procedure. This makes it easier to install CoCoALib on a wider range of computers. The C++ source code for CoCoALib has also been made cleaner and more portable.

CoCoALib is currently written using standard C++03; this does mean that compilation with a C++11 compiler may produce some harmless warnings about "deprecated" features (esp. `auto_ptr`). Now that C++ compilers which can handle the newer versions of C++ are widely available, we shall soon make a proper update to CoCoALib to use the newer C++ standards.

New functions in CoCoA-5

Here we mention some of the more significant mathematical additions. The website download page for CoCoA-5 includes a complete list of all the new functions and features; the same list can also obtained by running the function `RelNotes()` from inside CoCoA-5. Furthermore, a fully detailed description

of the work and decisions made for each software release may be found on the CoCoA *Redmine* website `https://cocoa.dima.unige.it/redmine`.

Most of the new functions added to CoCoA-5 have actually been added to CoCoALib, and then made available to CoCoA-5; the remaining few will be ported shortly.

- several efficient operations specifically for zero-dimensional ideals, mostly based on an efficient algorithm for computing minimal polynomials in zero-dimensional affine algebras (see *Minimal polynomials* below)
- more operations with algebraic extensions (even without a primitive element)
- faster conversion from a `string` to a `RingElem` — this is now the best way to read a large polynomial with very many terms.

Sectional matrix

The *sectional matrix* of an ideal I was introduced in [14] unifying the concepts of the Hilbert function of a homogeneous ideal I (along the rows) and its hyperplane sections (along the columns), and has recently been revived in [15]. Its theory and computation for a homogeneous ideal I is related to its `rgin`, (DegRevLex-gin).

```
/**/ use P ::= QQ[w,x,y,z];
/**/ I := ideal(x^4 -x*y^3,  x^2*(y -z),
                y^2*(y -z),  x*z^2 -y^3);
/**/ rgin(I);
ideal(w^3,  w^2*x,  w*x^2,  x^4,  x^3*y,  w^2*y^2,
      w*x*y^3,  x^2*y^3)
/**/ PrintSectionalMatrix(P/I);
  0  1   2   3   4   5   6
  -  -   -   -   -   -   -
  1  1   1   0   0   0   0
  1  2   3   1   0   0   0
  1  3   6   7   5   3   3
  1  4  10  17  22  25  28
```

The computation of `gin` and `rgin` in CoCoA, via a change of coordinates with random coefficients from a wide range, $[-10^6, 10^6]$, is based on the implementation of `RingTwinFloat` (see [3]).

Ideal of projective points

The efficient C code which was originally written for CoCoA-4 (see [7]) has now been made accessible from CoCoALib and CoCoA-5:

```
/**/ use P ::= QQ[x,y,z];
/**/ I := IdealOfProjectivePoints(P,
            matrix([[0,0,1],[1/2,1,1],[0,1,0]]));
/**/ I;
ideal(x*z +(-1/2)*y,  x*y +(-1/2)*y,
      x^2 +(-1/4)*y,  y^2*z -y)
```

Resolutions and graded Betti numbers

The current implementation for computing resolutions is rather naive, but will soon be improved — in this case porting the efficient but old C code for CoCoA-4 is impractical.

```
/**/ use R ::= QQ[x,y,z];
/**/ I := ideal(x^2+z^2,  x^2-y^2,  x*y-z^2);
/**/ resI := res(I);  PrintRes(resI);
0 --> R[-5] --> R[-3](+)R[-4]^2 --> R[-2]^3
```

A new addition is the ability to compute multigraded resolutions. There are functions to extract the Betti numbers in several formats (namely `BettiNumbers`, `BettiMatrix`, `BettiDiagram`, and functions to print them nicely: type `?betti` to see their manual pages)

```
/**/ -- first make a poly-ring with ZZ^2-grading
/**/ M := MakeTermOrd(matrix([[5,5,5,1,1],
                              [1,1,1,0,0] ]));
/**/ P := NewPolyRing(QQ, "t1,t2,t3,x,y", M, 2);
/**/ use P;
/**/ I := ideal(t1^6 -t3^6,  t1*t3*x^8 -t2^2*y^8);
/**/ resPI := res(P/I);  PrintRes(resPI);
0 --> R[-48, -8] --> R[-18, -2](+)R[-30, -6] --> R

/**/ PrintBettiNumbers(resPI);
----- 1 -----------
----- 2 -----------
  [18,  2]: 1
  [30,  6]: 1
----- 3 -----------
  [48,  8]: 1
--------------------
```

Gröbner Fan and Gfan library

There are several functions for accessing the operations of the `Gfan` library [18] (type `?gfan` for their manual pages). Using them A. Jensen implemented in CoCoA-5 a function for computing all distinct Gröbner bases:

```
/**/ use QQ[x,y,z];
/**/ I := ideal(x^3 +x*y -z,  x^2 -y*z);
/**/ GF := GroebnerFanIdeals(I);  indent(GF);
[
  ideal(x^2-y*z, x*y*z+x*y-z, y^2*z^2+y^2*z-x*z),
  ideal(x^2-y*z, x*z-y^2*z^2-y^2*z, y^3*z^2+...),
  ...
```

Since the Gröbner fan can sometimes be quite large, the function `GroebnerFanIdeals` will print a "⋆" as each new Gröbner basis is found if the verbosity level is 10 or higher.

Another problem with ideals having large Gröbner fans (with thousands of different orderings and bases) is that storing all the different ideals becomes impracticable. However, typically we are interested only in those bases satisfying a certain property. The function `CallOnGroebnerFanIdeals` calls a given function successively on each of the distinct Gröbner bases, without storing them in a huge list. This needs a little technical ability, but might make the difference between getting an answer or filling up the computer's memory.

Here is an example showing how to print out only those Gröbner bases comprising 3 elements (and the corresponding order matrix):

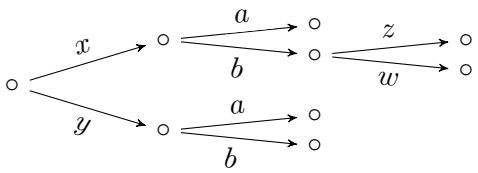
```
/**/ define GBOfLen3(I)
       if len(ReducedGBasis(I)) = 3 then
         println OrdMat(RingOf(I));
         println ReducedGBasis(I);
       endif;
     enddefine;
/**/ I := ideal(a^5+b^3+c^2-1,
                b^2+a^2+c-1,
                c^3+a^6+b^5-1);
/**/ CallOnGroebnerFanIdeals(I, GBOfLen3);
matrix(ZZ,
 [[1, 1, 1],
  [0, 0, -1],
  [0, -1, 0]])
[x^2 -y*z,  x*y*z +x*y -z,  y^2*z^2 +y^2*z -x*z]
....
```

Staged trees

A surprising application of monomial ideals in statistics is described in [17]. Given a polynomial whose power-products index all the cases in a data-set, find all *staged trees*

```
/**/ use QQ[a,b,x,y,z,w];
/**/ c_T := b*x*z +b*x*w +a*x +b*y +a*y;
/**/ trees := StagedTrees(c_T);
/**/ PrintTrees(trees);
-- number of trees = 2 ----------------------
-------- tree 1 -----------------------------
-- florets: [[x,  y],  [a,  b],  [z,  w]]
<
 x <
      a
      b <
           z
           w
 y <
      a
      b
-------- tree 2 -----------------------------
-- florets: [[a,  b],  [x,  y],  [z,  w]]
...
```

The first part is a textual representation for the tree



Knowing all such trees is a new key for the investigation of putative causal interpretations of datasets.

Hypersurface Implicitization

An efficient implementation of implicitization of hypersurfaces was recently added to CoCoALib. The CoCoA-5 interface has been made more user-friendly for rational parametrizations:

```
/**/ K := NewFractionField(RingQQt(1));
/**/ use K;
/**/ ParamDescr:=[ (1-t^2)/(1+t^2), 2*t/(1+t^2) ];
/**/ ImplicitHypersurface(ParamDescr);
x[1]^2 +x[2]^2 -1
```

Full details of the algorithms can be found in [8].

Minimal polynomials

Let $P = K[x_1, \ldots, x_n]$ be a polynomial ring with coefficients in a field K, and let I be a zero-dimensional ideal in P. It is well-known that the zero-dimensional affine K-algebra $R = P/I$ is a finite-dimensional K-vector space. Consequently, it is not surprising that minimal and characteristic polynomials from Linear Algebra can be successfully used to detect properties of R. For example, in the quotient ring $R = \mathbb{Q}[x, y]/(x^2, y^2)$ the minimal polynomial for the residue class of $x + y$ is z^3.

This point of view was taken systematically in the book [19] where the particular importance of minimal polynomials (rather greater than that of characteristic polynomials) emerged quite clearly. We studied the theory of minimal polynomials, developing efficient algorithms to compute them, and finding practical and efficient applications; here we recall the main results we described in [10] and in [20].

The first step was to implement in CoCoALib algorithms for computing the minimal polynomial of an element of R or that of a K-endomorphism of R. These functions are also accessible from CoCoA-5: type `?MinPolyQuot` for documentation.

Here is an example where we verify that the algebraic extension made by $\{\sqrt{2}, \sqrt[3]{3}, \sqrt[5]{5}, \sqrt[7]{7}\}$ is actually a field, and that $\sqrt{2} + \sqrt[3]{3} + \sqrt[5]{5} + \sqrt[7]{7}$ is a primitive element for the extension:

```
/**/ use P ::= QQ[a,b,c,d];
/**/ I := ideal(a^2-2, b^3-3, c^5-5, d^7-7);
/**/ IsMaximal(I); --> Is P/I is a field?
true
/**/ multiplicity(P/I);
210
/**/ MinPolyQuot(a+b+c+d, I, a);
a^210 -210*a^208 -210*a^207 + 21840*a^206 + ...
```

For computing minimal polynomials of elements of an algebra over $\mathbb{Q}$ we use an efficient modular approach and the rational reconstruction described in [4]. In particular, we dealt with the notion of *reduction of an ideal modulo p*, introducing the related notions of *ugly, usable, good and bad primes*. This is further investigated in a forthcoming paper [11].

As mentioned earlier, an efficient way of computing minimal polynomials lets us compute quickly a number of other important invariants of zero-dimensional affine K-algebras. For instance, in [10] we described algorithms for testing whether a zero-dimensional ideal is radical, maximal or primary, and others for computing the radical and the primary decomposition of a zero-dimensional ideal. In particular, it is noteworthy that in the case of coefficient fields of small finite characteristic, Frobenius spaces play a fundamental role, as described in [19, 20]. All the algorithms described have been implemented in CoCoA. Their efficiency is exhibited by the tables of timings in [10].

The computation of the primary decomposition of a zero-dimensional ideal computes minimal polynomials "behind the scenes":

```
/**/ use P ::= QQ[x,y,z];
/**/ I := ideal(3*y^2*z^2 +3*x*y^2 +1,
                x^4 +y*z^3,  y^4 +y*z^3)
/**/ PD := PrimaryDecomposition0(I);
/**/ len(PD);
9
```

The efficient computation of minimal polynomials is also a key ingredient in solving polynomial systems, and together with the other related algorithms we believe we can develop good tools for the SC^{2}community (see below).

News about SC^2

The CoCoA group is a member of the EU project "SC^2, Satisfiability Checking and Symbolic Computation" [1], so we are now collaborating with various members of the *Satisfiability Modulo Theory* community. In particular, we have built a first interface between CoCoALib

and the software MathSAT [16] in close collaboration with A. Griggio. There is also a separate collaboration with the developers of SMTRAT [13].

A simple example of calling the MathSAT function `msat_solve` from CoCoALib can be found in the example program `ex-MathSat1`. A more advanced interface, with dedicated CoCoALib C++ wrapper class for the data type `msat_env`, is part of the forthcoming CoCoALib 0.99560 distribution (September 2017).

The current interface permits CoCoA to use MathSat capabilities for solving linear inequalities; in the future the interface will be bi-directional allowing MathSAT to use CoCoA for tackling polynomial inequalities.

We illustrate the interface via an example. The linear inequalities are represented as matrices: each row representing one inequality. Consider the following system of inequalities:

$$\begin{cases} x_1 + 2x_2 + 3x_3 + 4 \leq 0 \\ 9x_1 + 8x_2 + 7x_3 \leq 0 \\ x_1 \neq 0 \\ x_1 + x_2 + 4 = 0 \\ x_2 < 0 \end{cases}$$

```
/**/ constraints :=
         record[ leq0 := matrix([ [1,2,3, 4],
                                  [9,8,7, 0] ]),
                 neq0 := matrix([ [1,0,0, 0] ]),
                 eq0 := RowMat([1,1,0, 4]),
                 lt0 := RowMat([0,1,0, 0])  ];
/**/ soln := MSatLinSolve(constraints);
/**/ GetCol(soln,1);
[-2, -2, -2/7]
```

This interface is still a "proof-of-concept" prototype; more MathSAT features will soon be added. The CoCoA-5 manual search `?MathSAT` will list all capabilities available in the current version.

Thanks

This research was partly supported by the project H2020-FETOPEN-2015-CSA_712689 of the European Union.
J. Abbott is an INdAM-COFUND Marie Curie Fellow.

References

[1] Erika Ábrahám, John Abbott, Bernd Becker, Anna M. Bigatti, Martin Brain, Bruno Buchberger, Alessandro Cimatti, James H. Davenport, Matthew England, Pascal Fontaine, Stephen Forrest, Alberto Griggio, Daniel Kroening, Werner M. Seiler, Thomas Sturm: SC^2*: Satisfiability checking meets symbolic computation (Project Paper)* LNCS 9791, pp. 28-43

[2] J. Abbott.: *The Design of CoCoALib* In N. Takayama, A. Iglesias (eds.) Proceedings of ICMS 2006, LNCS 4151:205–215. Springer (2006)

[3] J. Abbott: *Twin-Float Arithmetic* Journal of Symbolic Computation, in press: http://dx.doi.org/10.1016/j.jsc.2011.12.005 (2011)

[4] J. Abbott, *Fault-Tolerant Modular Reconstruction of Rational Numbers* J. Symb. Comp. 80P3, pp. 707–718, (2017).

[5] J. Abbott, A.M. Bigatti: *CoCoALib: a C++ library for doing Computations in Commutative Algebra* `http://cocoa.dima.unige.it/cocoalib/`

[6] J. Abbott, A.M. Bigatti, *CoCoA and CoCoALib: Fast prototyping and flexible C++ library for Computations in Commutative Algebra* Proc. 1st Workshop on Satisfiability Checking and Symbolic Computation, SC-Square 2016, CEUR Workshop Proceedings 1804, pp. 1–3, (2016).

[7] J. Abbott, A.M. Bigatti, M. Kreuzer, L. Robbiano, *Computing Ideals of Points*, Journal of Symbolic Computation 30, pp. 341–356 (2000).

[8] J. Abbott, A.M. Bigatti, L. Robbiano, *Implicitization of Hypersurfaces* Journal of Symbolic Computation 81, 20–40 (2017).

[9] J. Abbott, A.M. Bigatti, G. Lagorio: *CoCoA-5: a system for doing Computations in Commutative Algebra* `http://cocoa.dima.unige.it/`

[10] J. Abbott, A. Bigatti, E. Palezzato, L. Robbiano, *Computing and Using Minimal Polynomials* `arXiv:1704.03680`, (2017).

[11] J. Abbott, A. Bigatti, L. Robbiano, *Ideals modulo p* In preparation.

[12] J. Abbott, C. Fassino, M.L. Torrente: *Stable border basis for ideals of points* Journal of symbolic computation 43, pp. 883–894 (2008).

[13] E. Ábrahám *SMTRAT* Software available from `https://github.com/smtrat/smtrat/wiki`

[14] A. M. Bigatti, L. Robbiano. *Borel sets and sectional matrices* Ann. Comb., 1(1):197–213, (1997).

[15] A. M. Bigatti, E. Palezzato, M. Torielli. *Extremal behaviour in sectional matrices* `arXiv:1702.03292`, (2017).

[16] A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani: *The MathSAT5 smt solver* Proceedings TACAS 2013, volume 7795 of LNCS, pp. 93–107, (2013).

[17] C. Görgen, A.M. Bigatti, E. Riccomagno, J. Smith *Discovery of statistical equivalence classes using computer algebra* `arXiv:1705.09457` (2017).

[18] A.N. Jensen, *Gfan, a software system for Gröbner fans and tropical varieties*. Available from `http://home.math.au.dk/jensen/software/gfan/gfan.html`

[19] M. Kreuzer and L. Robbiano, *Computational Linear and Commutative Algebra* Springer, Heidelberg 2016.

[20] E. Palezzato, *Minimal Polynomials, Sectional Matrices, and Applications*; PhD thesis, Università degli Studi di Genova, (2017).