CoCoA and CoCoALib: Fast Prototyping and Flexible C++ library for Computations in Commutative Algebra

John Abbott
Institut für Mathematik
Universität Kassel
Kassel, Germany
http://www.dima.unige.it/∼abbott

Anna Maria Bigatti
Dipartimento di Matematica
Università degli Studi di Genova, Italy
http://www.dima.unige.it/∼bigatti

***Abstract*—The CoCoA project began in 1987, and conducts research into *Computational Commutative Algebra* (from which its name comes) with particular emphasis on Gröbner bases of ideals in multivariate polynomial rings, and related areas. A major output of the project is the CoCoA software, including the CoCoA-5 interactive system and the CoCoALib C++ library. The software is open-source (GPL v.3), and under continual, active development. We give a summary of the features of the software likely to be relevant to the SC-Square community.**

I. Introduction

CoCoA-5 is a well-established Computer Algebra System specialized in operations on polynomial ideals, and a number of allied operations (*e.g.* factorization, and linear algebra). It offers a dedicated, mathematician-friendly programming language, and numerous functions covering many aspects of Commutative Algebra. There is a strong emphasis on both rigour and ease of use. For details see [1].

The mathematical core of the software is CoCoALib. It is an open source C++ software library which has been designed to be user-friendly, facilitating integration with other software. The library is fully documented, and also comes with about 100 illustrative example programs (since it is often quicker to copy and modify existing working code than write it from scratch). For details see [2].

There is also a prototype "CoCoA server" giving access to many functions via a remote-procedure-call connection. Currently, communications to and from the server use an OpenMath-like syntax. A more sophisticated "remote session" communication model is envisaged, which will reduce overall transmission costs.

The CoCoA software is interesting to SC-square because it includes a well-documented, open source C++ library offering a good implementation of Gröbner bases, polynomial factorization, etc. The interactive system is well-suited to rapid prototyping, while the server offers possibilities for looser integration.

II. The Software Library CoCoALib

A crucial aspect of the CoCoA software is that it was designed from the outset to be an open-source software library (with two closely related applications being the interactive system and the compute server). This initial decision, together with the desire to help the software prosper, has many implications: *e.g.* designing a particularly clean interface for all functions with comprehensive documentation. This cleanliness makes it easy to integrate CoCoALib into other software in a trouble-free manner.

Many computer algebra systems have a two-tier structure: a compiled kernel with interpreter, and an external library of interpretable code for more advanced functions. Our aim with CoCoALib is different: we plan to have all functions implemented directly in the C++ library so that all features are available to programs which link to CoCoALib.

CoCoALib reports errors using C++ exceptions, while the library itself is exception-safe and (largely) thread-safe. The current source code follows the C++03 standard; a passage to the C++11 standard is planned for the near future (and this should enable the code to become fully thread-safe).

A. Library Design

We want CoCoALib to be a desirable software library, so it must be easy to use, reliable, robust and fast. Unfortunately, achieving all these features together is not always possible, so there are occasional compromises. One guiding principle is that we want **no nasty surprises**.

Here are the main features of the design of CoCoALib:

- it is well-documented, free and open source C++ code (under the GPL v.3 licence);
- the design is inspired by, and respects, the underlying mathematical structures;
- the source code is clean and portable;
- the user function interface is natural for mathematicians, and easy to memorize;
- execution speed is good with robust error detection.

Our design of CoCoALib aims to make it easy to write correct programs, and difficult to write incorrect ones or ones which produce "nasty surprises". While trying to follow this guideline we encountered some delicate aspects of the design:

- precise definition of a function's domain (*e.g.* what result should `IsPrime(-2)` give? And `IsPrime(0)`?)
- a choice between absolute mathematical correctness or decent computational speed (and a remote chance of a wrong answer)

In general, CoCoALib handles limit cases properly. The domain of each CoCoALib function is described in the documentation; for instance `IsPrime` throws an exception if the argument is not strictly positive — our reasoning is that it is unusual to want to test the primality of zero or a negative number, so it is likely that the routine which called `IsPrime` has already made a mistake, so it is better to report it "as soon as possible". CoCoALib offers the programmer the choice between absolute correctness or probable correctness and good speed, *e.g.* `IsPrime` and `IsProbPrime`.

An example of design: Finding a library interface which is easy to learn and use, mathematically correct, but also efficient at run-time often requires a delicate balance of compromise. We cite here one example from CoCoALib where the solution is untraditional but successful.

CoCoALib uses continued fractions internally in various algorithms. A *continued fraction* is an expression of the form:

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \frac{1}{a_3 + \cdots}}}$$

where a_0 is an integer, and $a_1, a_2, \ldots$ are positive integers. Every rational number has a finite continued fraction which, for compactness, is often represented as a list of integers $[a_0, a_1, a_2, \ldots, a_s]$.

The most natural implementation in CoCoALib would simply compute this list. But in many applications only the first few a_k are needed, and computing the entire list is needlessly costly. So the CoCoALib implementation produces an *iterator* (a basic concept well-known in object-oriented programming) which produces the values of the a_k one at a time.

B. Extending CoCoALib

Naturally most of the source code in CoCoALib was written by us, but the design of the library (and its openness) was chosen to facilitate and encourage "outsiders" to contribute. We distinguish two categories of contribution: code written specifically to become part of CoCoALib, and standalone code written without considering its integration into CoCoALib.

Specific Direct Contributions to CoCoALib: The first outside contribution came from M. Caboara, who wrote the code for computing Gröbner bases and related operations while CoCoALib was still quite young. At that stage the detailed implementation of CoCoALib was still quite fluid, and a number of pretty radical changes in the underlying data-structures were still to occur; yet despite these upheavals Caboara's implementation of Buchberger's algorithm required virtually no changes, thus confirming the good modularity and stability of the CoCoALib interface design.

Another significant outside contribution came from E. Saenz-de-Cabezon, who wrote the code for computing Mayer-Vietoris trees associated to monomial ideals. A significant aspect of this contribution is that the author worked completely independently relying entirely on the documentation of CoCoALib — thus confirming the quality of the documentation. His work has encouraged us to develop specialized, efficient handling for monomial ideals (see [7]).

A more recent contribution comes from M. Albert, who implemented an algorithm for computing Janet Bases of ideals in polynomial rings. Once a Janet Basis has been obtained, many ideal invariants can readily be determined ([4], [5]). Work is still under way to further expand this contribution.

C. Combining with External Libraries

We have combined some of the features of various external libraries into CoCoALib. An important step in each case is the "translation" of a mathematical value from its CoCoALib representation to that of the foreign library, and *vice versa*. To make it easier to do this CoCoALib offers operations for destructuring the various data-structures it operates upon.

One aspect of combining libraries which requires careful attention is any *global initialization* the libraries perform, for instance specifying the memory manager for the common underlying library GMP. CoCoALib addresses this by requiring explicit initialization of its globals; there are various options, including one for specifying a memory manager for GMP.

The first library we combined with CoCoALib is *Frobby* (see [8]) which is specialized for operations on monomial ideals. The experience also helped us improve the interfacibility of CoCoALib.

The most advanced integration we have achieved so far is with the *Normaliz* library (see [6]) for computing with affine monoids or rational cones. This is part of a closer collaboration which is described in more detail in [3]. In this particular case, a new data-structure (called `cone`) was added to CoCoALib to contain the type of value which Normaliz computes with.

The most recent integration was with *GFanLib* (see [9]) which is a C++ software library for computing Gröbner fans and tropical varieties. The experience gained from the earlier integrations made this a swift and painless operation.

There is also an experimental connection to some of the functions of GSL (GNU Scientific Library [10]). This is an interesting challenge because the interface has to handle two contrasting viewpoints: the exact world of CoCoALib, and the approximate (*i.e.* floating-point) world of GSL.

III. The Interactive System CoCoA-5

The CoCoA-5 system replaces the old CoCoA-4 system whose heritage can be traced back at least to 1989. For several reasons CoCoA-5 is a completely new implementation, whose design was developed under the precept that it be "as backward-compatible as (reasonably) possible" while also eliminating the limitations inherent in the older system.

All incarnations of the CoCoA system have been noted for their approachability especially for "computer-phobic" mathematicians, and CoCoA-5 is no exception. Indeed, the

aim was to make CoCoA-5 even friendlier than its forebears: a notable example is the importance given to generating genuinely helpful error messages.

Here is a typical error message from CoCoA-5; note that the error was actually signalled by CoCoALib as a C++ exception, which the interpreter caught, and then "translated" into human-readable form:

```
# X := 99 + FloorSqrt(-99);
--> ERROR: Value must be non-negative
--> X := 99 + FloorSqrt(-99);
-->             ^^^^^^^^^^^^^^
```

Note how the subexpression which actually triggered the error is indicated by "up-arrows", a most helpful feature in more complicated expressions.

Compared to earlier versions CoCoA-5 has a number of significant new abilities, notably including algebraic extensions and ring homomorphisms. The latter can be used to "move rigorously" values from one ring to another.

A. The Language in CoCoALib and in CoCoA-5

When designing CoCoALib and CoCoA-5 we envisaged researchers and advanced users wishing to tackle hard computations initially developing a prototype implementation in the convenient interpreted environment of CoCoA-5, and when the code is working properly, they translate it into C++ (using CoCoALib functions) for better performance. Consequently, one of the joint design goals of CoCoALib and the new CoCoA-5 language was to make it easy to convert CoCoA-5 code into C++ code built upon CoCoALib. That said, CoCoALib offers a richer programming environment, but also demands greater discipline from the programmer. For instance, CoCoA-5 does not have a special type for a "power-product" (it is just a polynomial with 1 term and coefficient 1), whereas CoCoALib has a special class, `PPMonoidElem`, which represents power-products. Thus a good translation into C++ of a CoCoA-5 program manipulating power-products will require some effort, but the reward should be a decisive gain in speed.

To facilitate the conversion into C++ we have, whenever possible, used the same function names in both CoCoA-5 and CoCoALib. We have also preferred traditional "functional" syntax in CoCoALib over object oriented "method dispatch" syntax, *e.g.* in CoCoALib we define `deg(f)` rather than `f.deg()`.

B. Extending CoCoA-5

The capabilities of CoCoALib and CoCoA-5 are continually expanding as the software evolves. So our design deliberately makes it easy to add new functions to CoCoA-5. In fact, there are several ways of extending CoCoA-5.

- The easiest way to add a new function is to write it in CoCoA-5 Language. Anyone can create new CoCoA-5 functions this way, and for instance give them to students or colleagues.
- Often there are several functions to be added together; in this case it is best to place them all in a **CoCoA-5 package**. The package indicates which of those functions it will *export*; the rest are internal auxiliary functions.
- The last way is to write the new functions in C++, add them to CoCoALib, and then make them "visible" to CoCoA-5. This last operation is normally quite straightforward thanks to an ingenious combination of C++ inheritance and C macros (see [3]). Indeed we use exactly this mechanism for making standard CoCoALib functions accessible from an interactive CoCoA-5 session.

While many CoCoA-5 functions just call CoCoALib directly, there are still a few packages containing functions which have yet to be migrated into CoCoALib.

IV. The CoCoAServer

Included in the CoCoA software distribution is the *CoCoAServer*, currently still a prototype which provides a client/server mechanism for accessing the capabilities of CoCoALib. It uses an OpenMath-like language to accept computation requests, and then send the result back. An early use of the protype was to grant access to CoCoALib features from CoCoA-4.7 while the new CoCoA-5 system was under development.

The advantages of computing via a "server" are that it can be called by any other "client" software (which has an OpenMath interface), and it avoids the delicate intricacies of close integration occuring in monolithic compilation.

Currently the server remains in prototype form as resources are directed, for the time being, at CoCoALib and CoCoA-5.

V. Conclusion

The CoCoA software offers access to advanced capabilities in computational commutative algebra via three interfaces: an interactive system, a C++ library, and a prototype server. This variety allows users to choose whichever approach suits them best.

References

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

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

[3] J. Abbott, A.M. Bigatti, C. Söger, *Integration of libnormaliz in CoCoALib and CoCoA 5* Proc. ICMS 2014, Springer LNCS 8592, pp. 647–653, 2014.

[4] M. Albert, *Janet Bases in CoCoA*; bachelor thesis, Institut für Mathematik, Universität Kassel, 2011.

[5] M. Albert, *Computing Minimal Free Resolutions of Polynomial Ideals with Pommaret Bases* master thesis, Institut für Mathematik, Universität Kassel, 2013.

[6] W. Bruns, B. Ichim, T. Römer, C. Söger *Normaliz*. Available from `http://www.mathematik.uni-osnabrueck.de/normaliz`

[7] O. Fernàndez-Ramos, E. García-Llorente, E. Sáenz-de-Cabezón *A monomial week* (Spanish) Gac. R. Soc. Mat. Esp. 13, No. 3, 515–524, 2010.

[8] B.H. Roune, *Frobby*. Available from website `http://www.broune.com/frobby`

[9] A.N. Jensen, *GFan*. Available from website `http://home.math.au.dk/jensen/software/gfan/gfan`

[10] M. Galassi *et al.*, GNU Scientific Library Reference Manual (3rd Ed.), ISBN 0954612078. *GSL* complete package available from website `http://www.gnu.org/software/gsl`