CoCoA: Computations in Commutative Algebra

John A. Abbott, Anna M. Bigatti*

Dip. di Matematica, Univ. di Genova, Via Dodecaneso 35 - 16146 Genova, Italy
[cocoa@dima.unige.it]

2000 Mathematics Subject Classification. 13P05, 13P10, 13P99, 68W30

What is CoCoA?

CoCoA is a special-purpose system for doing Computations in Commutative Algebra. It runs on all common platforms. ([1])

CoCoA is one of the products of a research team in Computer Algebra based in Genoa (Italy) with the cooperation of researchers and students in Europe and North America.

CoCoA is freely available software for research and educational purposes: the latest version is CoCoA 4.6 (May 2006) which can be obtained from http://cocoa.dima.unige.it

as well as from the mirrors sites

http://www.reed.edu/mirrors/cocoa

The system offers a textual interface, an Emacs mode, and a graphical user interface common to most platforms.

The main features of CoCoA

An important purpose of the CoCoA program is to provide a “laboratory” for studying computational commutative algebra: it together with Singular and Macaulay 2 form an elite group of highly specialized systems having as their main forte the capability to calculate Gröbner bases. Although a number of general purpose symbolic computation systems (e.g. REDUCE and Maple) do offer the possibility to compute Gröbner bases, their non-specialist nature implies a number of severe compromises which make them far less suitable to act as a laboratory: e.g. relatively poor execution speed and limited control over the algorithm parameters.

Aside from computing Gröbner bases CoCoA’s particular strengths include ideal/module operations (such as syzygies and minimal free resolutions, intersections, divisions, the radical of an ideal, . . . ), polynomial factorization, exact linear algebra, computing Hilbert functions, and computing with zero-dimensional schemes and toric ideals.

The usefulness of these technical skills is enhanced by the mathematically natural language for describing computations. This language is readily learned by students, and enables researchers to explore and develop new algorithms without the administrative tedium necessary when using “low-level” languages.

The users of CoCoA

Currently CoCoA is used by researchers in several countries. Most of them are Commutative Algebraists and Algebraic Geometers, but also people working
in different areas such as Analysis and Statistics have already benefitted from our system.

CoCoA places great emphasis on being easy and natural to use, so it is also used as the main system for teaching advanced courses in several universities in Europe and North America. It is mentioned in some of the most widely used text books in Computational Algebra, and plays a major role in the book Krenzer-Robbiano “Computational Commutative Algebra” (Springer).

The future of CoCoA

Lately the CoCoA project has entered a new phase: completely rebuilding and restructuring the program employing more modern algorithms where available. The aims of the new version include offering better flexibility and performance while retaining the simplicity of use for which CoCoA has become widely appreciated.

A crucial design decision was the passage from C to C++ as the implementation language: the improved expressivity of C++ allows the source code to be more readable while offering better run-time performance. We expect concomitant benefits for future maintenance of the source. The new design is expressly as a library; a server (communicating via OpenMath) and a standalone interactive system will be built on top of this library.

CoCoALib, being the core of the project, is also its most evolved part, and is the part that we shall look at most closely. In keeping with the theme of ready accessibility the software is to be free and open in the sense of the GPL.

Philosophy of CoCoALib

Our aim is for CoCoALib to offer reference implementations of the principal algorithms in computational commutative algebra. The development of the library and the other components requires an enormous investment of time and resources. So that this investment is worthwhile we want to ensure that the software is widely available and will live for a long period of time. Consequently our implementations have to satisfy various design criteria:

* the code must exhibit good run-time performance
* the source code must be clear and well designed
* the source code must be well documented (both for users and maintainers)
* the source code must be clean and portable
* the code must be easy and natural to use

One implication of these criteria is that the design should reflect the underlying mathematical structure since this will ensure that the library is natural to use.

Another implication is that we regard clear and comprehensible code as being generally more desirable than arcane convoluted code striving to achieve the utmost in run-time performance because clear code is easier to maintain and should live longer. Our experience is showing that this emphasis on cleanliness is also providing quite good run-time performance.