# New Flavours of CoCoA

**John Abbott, Anna M. Bigatti**
**(Università di Genova)**

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

*There are two ways of constructing a software design.*
*One way is to make it so simple*
*that there are obviously no deficiencies.*
*And the other way is to make it so complicated*
*that there are no obvious deficiencies.*
*The first method is far more difficult.*
C. A. R. Hoare


## What is CoCoA?

First released in 1988, CoCoA is a *special-purpose* system for doing **Co**mputations in **Co**mmutative **A**lgebra: *i.e.* it is an interactive system specialized in the algorithmic treatment of polynomials, and is *freely available* for most common platforms.

One of the main purposes of the CoCoA system is to provide a virtual "laboratory" for using and studying computational commutative algebra: it belongs to an elite group of highly specialized systems having as their main forte the capability to calculate Gröbner bases. This means that CoCoA is optimized for computing with multivariate polynomials, their ideals and modules. Other special strengths of CoCoA include polynomial factorization, exact linear algebra, Hilbert functions, zero-dimensional schemes, and toric ideals.

The usefulness of these technical skills is enhanced by its programming language, the CoCoALanguage, which places great emphasis on being easy and natural to use. Consequently, CoCoA is the system of choice both for teaching advanced courses in several universities, and also for many researchers wanting to explore and develop new algorithms without the administrative tedium necessary when using "low-level" languages.

## The new Flavours of CoCoA

About 10 years ago, a new initiative began: namely, to rebuild the CoCoA software laboratory while removing the inherent limitations of the original. The new software comprises three main components: a C++ library (CoCoALib), an algebra computation server (CoCoAServer), and an interactive system (CoCoA-5). Of these components CoCoALib is the heart; it embodies all the "mathematical knowledge" and it is currently the most evolved part ([3]). The role of the other two parts is to make CoCoALib's capabilities more easily accessible.

The first result of this approach is the collaboration with the project ApCoCoA (Applied Computations in Commutative Algebra), lead by Martin Kreuzer in Passau, which is built upon CoCoA and CoCoALib. It applies both symbolic and numerical computation to tackle "real world" problems.

All the new code is free and open source software. It is downloadable from our website ([3]) and released under GPL.

### CoCoALanguage

The design and development of the new mathematical core of CoCoA was conducted by mathematicians (the authors of this paper); in contrast the design of the language, parser, and interpreter was ably assisted by Giovanni Lagorio, computer scientist and expert in languages — a most welcome addition to the CoCoA team!

Our main challenge was to design a more powerful language while keeping it as compatible as possible with the language in CoCoA-4.

In the new language rings and functions are "normal objects", more properly called "first class values", and can be assigned and passed as arguments, thus avoiding some of the convolutions needed in CoCoA-4.

We decided to remove certain "features" which often led to ambiguities, frustration for beginners, or hidden bugs (code working with no error messages, but

doing the wrong thing). One of the nice "features" of CoCoA was letting the user write `A B` as a shortcut for `A*B`. This had a number of unfortunate implications; perhaps the most painful was that a forgotten "`;`" would become a multiplication with unexpected results!

So we came to the hard decision to require always "`*`" for multiplication. But do not worry! Where the old way is truly handy, *e.g.* when writing power-products in polynomials, we offer the `*** ... ***` shortcut inside which the "old rules" apply:

```
I := *** Ideal(2x^2y-z, 3xz-5yz^3) ***;
```

### Improved Errors!

The new design of the CoCoALanguage and the careful implementation of the parser and interpreter allow far superior handling of errors.

All CoCoA-4 users have surely encountered an unhelpful error message like this:

```
ERROR: parse error in line 21
```

The new error messages will tell you what and where the error was:

```
If N>1; Then F := (x-1)^N; Else ....

ERROR:
I was expecting "Then" but I've found ";"
If N>1; Then F := (x-1)^N; Else
        ^
```

Also for errors during evaluation the messages are equally informative. Moreover, as the true error often does not lie in the function which signalled it but in the function which called it (or in the function which called that one, or ...), the new error messages include the list of the location of the calls in the stack.

So when we ask the users who have already switched to CoCoA-5 for their comments they say: "CoCoA's new errors are really great!" ;-)

### Mathematical Foundations

The greatest change in the "mathematical part" comes from the fact that the new core (CoCoALib) was designed by a mathematician: the internal data types are carefully implemented to reflect their mathematical underpinnings.

In particular there is a greater variety of ring constructors (see below in Section CoCoALib), and any commutative ring can be used as the ring of coefficients of polynomial rings.

In parallel with this flexibility in ring construction CoCoA-5 offers proper ring homomorphisms to map ring elements (`RingElem`) between rings.

Both rings and ring homomorphisms are "first class values", so they can be assigned and passed as arguments. Ring homomorphisms can be called as functions and can be composed, as we see here:

```
R ::= QQ[a,b];
K := NewFractionField(R);
Use P ::= K[x,y,z];

G := (1/a)*x + 2*y;
LC(G);  -- leading coefficient
1/a
LT(G);  -- leading term
x
Use S ::= K[x];
phi := PolyAlgebraHom(P, S, [x,1,0]);
phi(G);
(1/a)*x +2
psi := PolyAlgebraHom(S, K, [100]);
f := psi(phi);  -- composition
f(G);
(2*a +100)/a
```

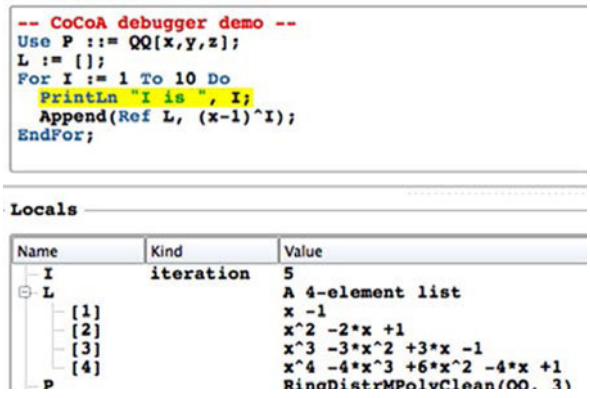
### User Interfaces

The CoCoA system provides three user interfaces: plain text, Emacs, and a custom graphical user interface (abbr. GUI).

Fans of the Emacs interface (as we are) will be pleased to know that "Nothing's changed"! Apart from a handful of small fixes and improvements — it is difficult to improve what is already almost perfect ;-)

The new GUI is still under development; it already boasts several improvements and interesting new features. The new input editor has coloured syntax and an indicator for balanced parenthesis: very useful against silly typos! A handy new feature is the "Reported Location" menu, which will take you to the exact spot where the errors were signalled.

Another special feature (alas! not available in the Emacs interface) is the debugger window in which the user can execute the CoCoA code step by step while monitoring the values of the variables.




### Adding Functions to CoCoA

We are not alone in developing CoCoA! Certainly, we are constantly adding new features; but we are also delighted when external contributors give us software donations. So, both for ourselves and for CoCoA users /contributors, we have made it easier to add new functions to CoCoA.

In CoCoA-4 the usual way to extend its capabilities was to encapsulate your CoCoALanguage functions in a `Package` which can then be passed to your colleagues (or, better, to the CoCoA authors!) We have simplified this process in CoCoA-5.

One guiding principle when we designed the new CoCoALanguage was to ensure that well-written CoCoA-4 packages would need little or no change to work in CoCoA-5. Indeed many capabilities of CoCoA-4 were implemented as packages. Porting these packages to CoCoA-5 was just an easy stepping stone towards complete integration where the same capabilities will be fully integrated into CoCoALib (with anticipated improvements in speed).

In the new CoCoA there is a second alternative for adding new features: implement in C++ (as part of CoCoALib) then adjoin the new functions to CoCoA-5. A key point in the design of the new CoCoA interpreter was to facilitate the adjunction of CoCoALib functions to CoCoA-5, so they become readily accessible also from CoCoALanguage. After having added hundreds of functions this way, we can safely say it is almost as easy as writing new implementations in CoCoALanguage.

## CoCoALib

The mathematical core of CoCoA-5 is the C++ library CoCoALib whose aims include offering better flexibility and performance while retaining the simplicity of use for which CoCoA has become widely appreciated.

CoCoALib is unique in its field because, right from the outset, it was designed as an open source library satisfying various design criteria:

- be easy and natural to use
- exhibit good run-time performance
- have a firm mathematical basis (following books [9, 10])
- be clear and well designed
- be clean and portable
- be well documented (both for users and maintainers)

This makes it an ideal choice as a basis upon which other researchers can develop efficient and robust implementations of their algorithms. Naturally we hope that many of these implementations will then be donated as new components for the library, helping to expand it.

We regard clear and comprehensible code as being generally more desirable than convoluted code striving for the highest possible speed. Conveniently, our experience up to now shows that this emphasis on cleanliness is also providing quite good run-time performance! In particular Gröbner bases computations are much faster than in the old CoCoA-4 software (written in C), and are now aligned with the other specialized systems Macaulay and Singular.

The inheritance mechanism of C++ plays a crucial role in the design of CoCoALib (see [1]), especially in the challenge of reconciling the traditionally conflicting goals of (mathematical) abstraction and efficiency: for example it is used to express the mathematical relationships between the various sorts of rings and their specific functions (*e.g.* `deg` for polynomial rings, `den` for fraction fields)

Being well aware that the usefulness of software is critically dependent on its documentation, we offer extensive documentation aimed both at guiding users and at aiding maintainers and contributors. And being even more aware that no one likes to read documentation, we also offer a good selection of example programs — so you can just cut-and-paste rather than read through the documentation!

### Approximately...

CoCoALib is primarily concerned with computations in Commutative Algebra, and therefore with *exact* computations on polynomials, nevertheless it also offers some facilities for exploring the world of approximate algebra (see the book [6]). Two complementary approaches are: using *approximate computations* to solve *exact problems*, and applying Commutative Algebra "exact techniques" to solve *approximate problems*.

### Twin-Float Arithmetic

A facility which CoCoALib offers for the first approach is twin-float arithmetic which can be used as a (generally) faster substitute for exact rational arithmetic with a heuristic guarantee of correctness. For full details see [2]; here we give just a brief intuitive outline.

Before computation begins, the user specifies a minimum acceptable accuracy. Then every (exact) rational input is converted into a *twin-float*: *i.e.* a *high precision* floating point value together with a *heuristic estimate* of the accuracy.

Every arithmetic operation on twin-float values checks that the heuristically estimated accuracy of the result is sufficient; if not, the operation fails. If no arithmetic step in our computation has failed the computation finishes and the result is (heuristically) guaranteed, otherwise we have to restart the entire computation specifying a higher precision.

Another special feature of twin-floats is the ability to recover a rational number from a twin-float value. This capability permits the recovery of the exact rational answer from a twin-float result under suitable circumstances; *e.g.* an exact Gröbner basis can be obtained from one computed using twin-floats.

The implementation in CoCoALib is as a `ring`, named `RingTwinFloat`, making it easy to use this arithmetic in a wide range of applications.

### Approximate Border Bases

Given a set of *exact* points CoCoA can compute quickly the reduced Gröbner basis of the ideal of the polynomials vanishing at those points. But, when the points are

measurements coming from the *real world* then their coordinates are known only approximately.

In this approximate context, the notion of Gröbner basis, which is so important in exact commutative algebra, can exhibit a fatal weakness: it can be structurally unstable in the presence of infinitesimal perturbations. It has recently been shown that in the zero-dimensional case these problems of structural instability can be eliminated by using instead a Border Basis.

Together with C. Fassino and M.-L. Torrente we have developed the notions and theory necessary to apply the Buchberger-Möller algorithm to approximate points, and a robust prototype implementation is included in CoCoALib (see [5], [7]).

This is a rapidly developing topic, promising to be of definite interest to various aspects of both theoretical and practical research.

# References


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

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

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

[4] J. Abbott, A.M. Bigatti, G. Lagorio: CoCoA-5.0: a system for doing Computations in Commutative Algebra `http://cocoa.dima.unige.it/`

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

[6] J. Abbott, L. Robbiano, (eds.): Approximate Commutative Algebra, Springer (2009)

[7] Fassino, C.: Almost Vanishing Polynomials for Sets of Limited Precision Points, Journal of symbolic computation 45, 19–37 (2010)

[8] M. Kreuzer *et al.* The ApCoCoA Project, `http://www.apcocoa.org/`

[9] M. Kreuzer, L. Robbiano: Computational Commutative Algebra 1, Springer, Heidelberg (2000, 2008)

[10] M. Kreuzer, L. Robbiano: Computational Commutative Algebra 2, Springer, Heidelberg (2005)