CoCoA and CoCoALib: Gröbner bases for everyone
John Abbott & Anna M. Bigatti – Università di Genova, Italy

Current Trends on Gröbner Bases
July 2015, Osaka
What is this talk about?

This talk is about software!
I’ll try to give... *a Gröbner basis to everyone*

The audience in this room is now thinking...

- “What is CoCoA?”
- “I don’t care: I prefer Macaulay2 / Singular / Risa/Asir / Maple / ...”
- “I’m still using the old CoCoA (version 4)”
- “What’s new in the new version of CoCoA-5?”
- “What’s the licence of CoCoALib? Can I link it with my program?”
- “What can be done with CoCoALib and not in CoCoA-5?”
- ....
What is CoCoA?

History
The CoCoA project started in 1987 under the lead of L. Robbiano (Giovini & Niesi & Capani)
**Aim:** a *mathematician-friendly* software for

Computations in Commutative Algebra

especially **Gröbner bases.**

Present
It has evolved and has been rewritten, always maintaining this tradition, and now offers:  (CoCoA-4, CoCoA-5: Abbott & Bigatti)

- an open source **C++ software library:** CoCoALib
- a **new interactive system:** CoCoA-5
- a prototype OpenMath-based **server**
The easiest way to compute a Gröbner basis is with the interactive system CoCoA-5. A small sample:

```
Use QQ[x,y,z];
I := ideal(x^3 + x*y^2 - 2*z, ..... , ..... , ..... );
GBasis(I);  // same as "print GBasis(I);"
```

The CoCoA language: no declaration of variables (dynamically typed)

```
A := (x-y)^3 * (z-2);  // A is now a RINGELEM
factor(A);

A := mat([[x,y,z], [1,2,3], [0,x-1,1/2]]);  // A is now a MAT
det(A);

A := ideal(x^3 + x*y^2, (x-1)^3, y*z-z^2);  // A is now an IDEAL
SyzOfGens(A);
```
What can I compute with CoCoA?

- Gröbner bases of ideals/modules, wide choice of term orderings
- special handling for ideals of points
- special handling for monomial ideals
- Hilbert series, resolutions, Betti numbers
- polynomial factorization
- basic exact linear algebra (LinSolve, LinKer, eigenvectors, det)
- approximate points: border bases, polynomial relations
From CoCoA-4 to CoCoA-5: new mathematics!

More rings: algebraic extensions, fraction fields ...

```plaintext
use R ::= QQ[a];  // "use" "::=" for special ring syntax: QQ[x] vs L[3]
K1 := NewFractionField(R);  // K1 is QQ(a)
K2 := NewQuotientRing(R, ideal(a^2-2));  // K2 is QQ[a]/(a^2-2)
use P ::= K1[x,y,z];
f := x - (1/a)*x + y;  // viewed as ((a -1)/a)*x +y in P
```

Ring homomorphisms

```plaintext
phi := CanonicalHom(R, K1);  // phi: QQ[a] --> QQ(a)
psi := CanonicalHom(R, K2);  // psi: QQ[a] --> QQ[a]/(a^2-2)
theta := CanonicalHom(K1,P) (phi);  // theta: QQ[a] --> QQ(a)[x,y,z]
```

```plaintext
use R;  // polynomials are read as elements in R = QQ[a]
1/phi(a^2 + 2*a -1);  // gives 1/(a^2 +2*a -1) in K1
1/psi(a^2 + 2*a -1);  // gives ((2/7)*a -1/7) in K2
1/theta(a^2 + 2*a -1);  // gives 1/(a^2 +2*a -1) in P
```
New in the **CoCoA-5** language

Even though all new, more robust and expressible **CoCoA-5** language is mostly compatible with CoCoA-4

“**Invisible multiplication**” \((xy\) for \(x*y\)) gave CoCoA-4 many constraints. Now allowed only inside **triple-**:

\[
\begin{align*}
I &:= \text{ideal}(2x^2y - z, 3xz - 5yz^3); \\
I &:= *** \text{Ideal}(2x^2y - z, 3xz - 5yz^3) ***;
\end{align*}
\]

**Rings and functions** are now **first class values**: can be assigned and passed as arguments

Better errors! I mean **error messages!!! ;-)** make it easy to learn and to update CoCoA-4 code

*(Demo)*
New in the CoCoA-5 language

Even though all new, more robust and expressible CoCoA-5 language is mostly compatible with CoCoA-4.

“Invisible multiplication” (xy for x*y) gave CoCoA-4 many constraints. Now allowed only inside triple-∗:

\[
\begin{align*}
I := \text{ideal}(2x^2y - z, 3xz - 5yz^3); \\
I := *** \text{Ideal}(2x^2y - z, 3xz - 5yz^3) ***;
\end{align*}
\]

*Rings and functions* are now *first class values*: can be assigned and passed as arguments

Better errors! I mean *error messages*!!! ;-) make it easy to learn and to update CoCoA-4 code

*(Demo)*
More than CoCoA itself

Several ways of extending CoCoA-5 for your needs

- Write your own functions in CoCoA-5 language

```coconut
define StrangeFunction(X)
    if type(X) = INT then return 2^X;
    elif type(X) = MAT then return det(X);
    endif;
    return X;
enddefine;
```

- Collect some functions into a new CoCoA-5 package

- Write the new functions in C++ inside CoCoALib, and then make them “visible” to CoCoA-5 (the new interpreter makes this last step really easy!)
Do you have a promising prototype of your algorithm in CoCoA-5 and you want to translate it into C++ for better performance? CoCoALib!

To facilitate this conversion CoCoALib we used:

- same function names as in CoCoA-5 (whenever possible)
- functional syntax e.g. GBasis(I) rather than object-oriented method call I.GBasis()

Use QQ[x,y,z];
I := ideal(x^3 + x*y^2 - 2*z, ...., ...., .... );
GBasis(I); // same as "print GBasis(I);"

This small sample literally translates into:

```cpp
ring P = NewPolyRing(RingQQ(), symbols("x", "y", "z"));
ideal I = ideal(ReadExpr(P, "x^3 + x*y^2 - 2*z"), ........ );
cout << GBasis(I);```

use R ::= QQ[a];
K2 := NewQuotientRing(R, ideal(a^2-2)); // K2 is QQ[a]/(a^2-2)
psi := CanonicalHom(R, K2); // psi: QQ[a] --> QQ[a]/(a^2-2)

use R; // polynomials are read as elements in R = QQ[a]
f := 1/psi(a^2 + 2*a -1); // gives ((2/7)*a -1/7) in K2
f;

This sample literally translates into:

```
ring R = NewPolyRing(RingQQ(), symbols("a"));
rings K2 = NewQuotientRing(R, ideal(ReadExpr(R, "a^2-2")));
RingHom psi = CanonicalHom(R, K2);

RingElem f = 1/psi(ReadExpr(R, "a^2 + 2*a -1"));
cout << "f is " << f << endl;
```
CoCoALib: the C++ mathematical brain of CoCoA-5

- (aim) all CoCoA-5 functionalities available in CoCoALib
- CoCoA-5 interpreter: easy to expose CoCoALib functions

CoCoALib: C++ library

- Designed to be easy to use
- Execution speed is good
- Well-documented, including many examples programs

- Free and open source C++ code (GPL3 licence)
- Source code is clean and portable (C++03)
- Design respects the underlying mathematical structures (inheritance, no templates)
- Robust (Motto: “No nasty surprises”), exception-safe, thread-safe
Design goal of the CoCoA-5 interpreter

easy to expose CoCoALib functions to CoCoA-5

Best example: “One-liner”

The function JanetBasis expects an ideal
(and outputs a list of polynomials)

The code to expose it to CoCoA-5 is just one line (C macro)

```
DECLARE_COCOALIB_FUNCTION1(JanetBasis, IDEAL)
```

meaning: 1 argument of type IDEAL (wrapper for CoCoALib ideal)

Output type is automatically determined and wrapped up for CoCoA-5
## Gröbner bases and beyond: contributions

### Authors

**CoCoALib** (John Abbott & Anna Bigatti)  
Parser and interpreter for **CoCoA-5** (Giovanni Lagorio)

But the openness and clean design of the library was chosen to encourage contributions

### Direct contributions to **CoCoALib**

- Mathematical support and feedback (L. Robbiano)
- Gröbner bases structure, ideal/module operations (M. Caboara)
- Mayer-Vietoris trees (E. Sáenz de Cabezón)
- Janet and Pommaret Bases (M. Albert and W. Seiler)
- Approximate points (M. L. Torrente and C. Fassino)
External libraries integrated with CoCoALib

- B. Roune: *Frobbby* (monomial ideals)
- C. Söger: *Normaliz* (affine monoids or rational cones)
- A. N. Jensen: *GFan* (Gröbner fans and tropical varieties) – soon

experimental interface with *GSL* (GNU Scientific Library)

Example: *libnormaliz*

One file: *AlgebraicCore/ExternalLibs-Normaliz.C* (and .H)
- definition of the (CoCoALib) class *cone*
- functions for data conversions between the two libraries
- functions actually available to the CoCoALib user

Utilities: *tests/test-normaliz1.C* and *examples/ex-normaliz1.C*

```bash
./configure --with-libnormaliz=PATH-TO/libnormaliz.a
make
```
Gröbner bases and beyond: contributions

Contributions

...last but not least:

feedback from CoCoA-5 and CoCoALib users!
cocoa@dima.unige.it
http://cocoa.dima.unige.it

I hope there were Gröbner bases for everyone ;-)  

Thank you!