CoCoALib

a C++ library for Computations in Commutative Algebra

John Abbott
Università di Genova, Italy

CoCoA and CoCoALib

The CoCoA is a beautiful car and you can drive it where you want...

... but if you need speed and power you need to be an expert driver and “talk” with the engine: CoCoALib.

CoCoALib has been designed to be an open source C++ library

in other words

to be used, compiled, and extended by everyone, not just the authors.

Design Philosophy behind CoCoALib

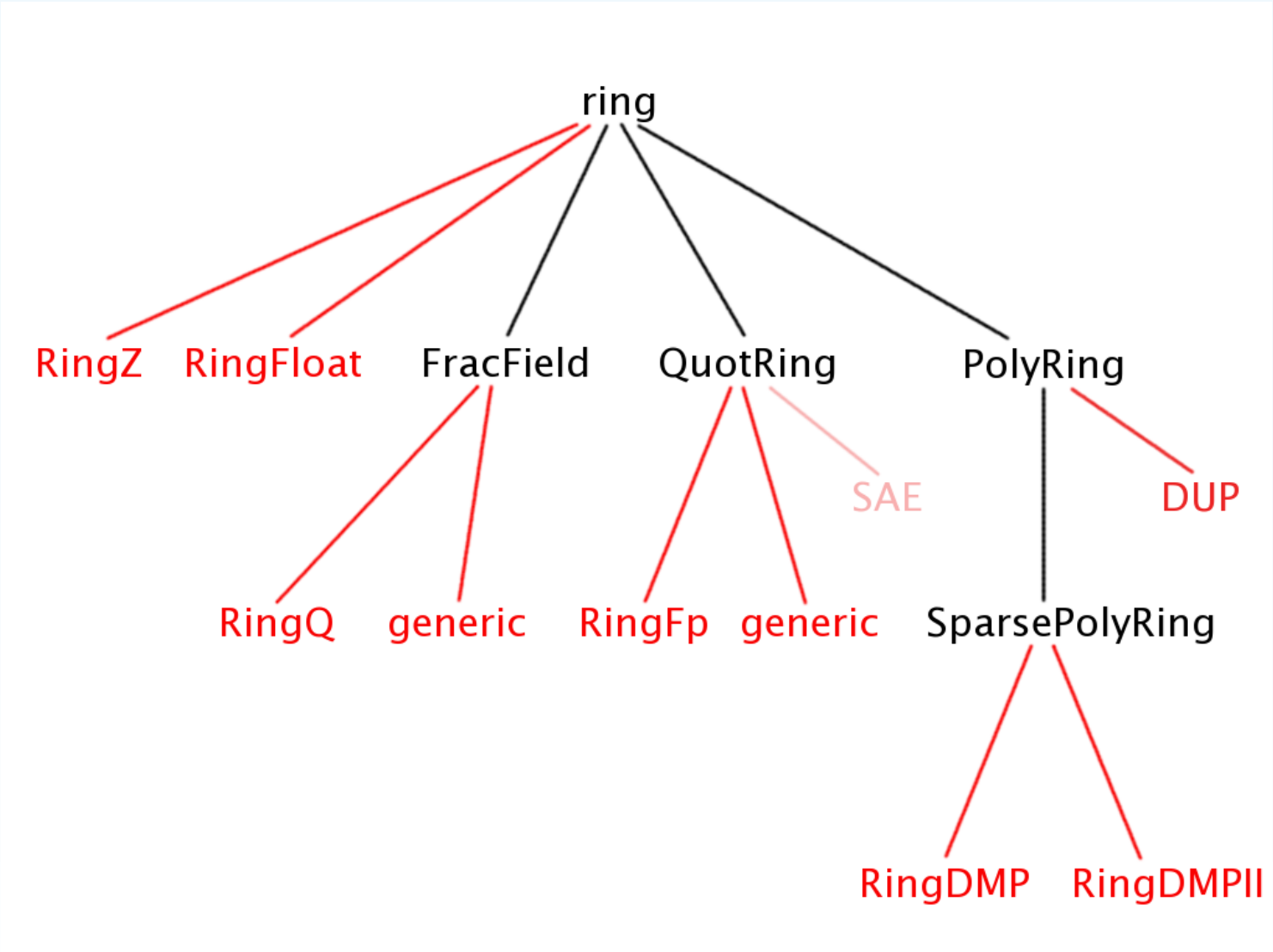
Basic goals of the design: the code must...

- be **easy and natural** to use
- have **firm mathematical basis** (Kreuzer-Robbiano book)
- exhibit **good run-time performance**
- be **well documented** (for users & maintainers)
- be clean and **portable**

Just a few obstacles from C++

- we cannot write `2/3`: interpreted as integer division $\longrightarrow$ 0
  `QQ(2,3)`
- we cannot write `x*y^4`: problems with operator priority $\longrightarrow$ `(x*y)^4`
  `x*power(y,4)`
- C++ lists/vectors are a bit unfriendly for C++ beginners

Ring Inheritance Diagram



(Run *examples/ex-PolyRing1.C*)

Clean vs Efficient

- **For all users** natural syntax with extensive checking
  `a = b + c;`
- **For experienced users** syntax for faster unchecked operations
  `R->myAdd(rawa, rawb, rawc);`
- **For developers** there are several debugging aids
  `MemPool`

General rule: use the clean syntax!

If you know how to profile (`gprof`) you will see how many times any function is called, and you then decide if it is worth using the faster and unchecked call.

God invented the integers...

Two ways to represent integers: **ZZ** and **RingZ**

```
ZZ three = ZZ(3);
ZZ seven = ZZ(7);
cout << seven/three;      // OK    value = 2
cout << -seven/three;     // OK    value = -3
cout << seven/(-three);   // FAILS

RingElem three = RingElem(RingZ(),3);
RingElem seven = RingElem(RingZ(),7);
cout << seven/three;      // FAILS
cout << -seven/three;     // FAILS
cout << seven/(-three);   // FAILS
```

Two ways to represent rationals: **QQ** and **RingQ**

```
QQ SevenThirds = QQ(7,3);
cout << SevenThirds + 2/3;     // Nasty surprise!!
cout << SevenThirds + QQ(2,3);// OK
```

Matrices

Reading and **assigning** entries:

```
M(i,j) ... \\  read access
SetEntry(M, i, j, ...); \\ assignment
```

A matrix **view**:

```
ConstMatrixView Id40000 = IdentityMat(R, 40000);

MatrixView TrM = transpose(M);
SetEntry(TrM, i,j, 123); \\ modifies M(j,i)
```

And also `submat`, `ColMat`, `RowMat`, `DiagMat`, `BlockMat`, `ConcatHor`, ...

Making a **new** matrix:

```
matrix TrM = NewDenseMat(transpose(M));
```

Coding conventions

- single words: lower case ideal, indet, coeff, ring,..
- more words: CamelCase RingElem, PolyRing, ...
- returning boolean: Is + CamelCase IsEmpty, IsProbPrime, IsDivisible, ..
- member functions: my + CamelCase myAdd, myLen,...

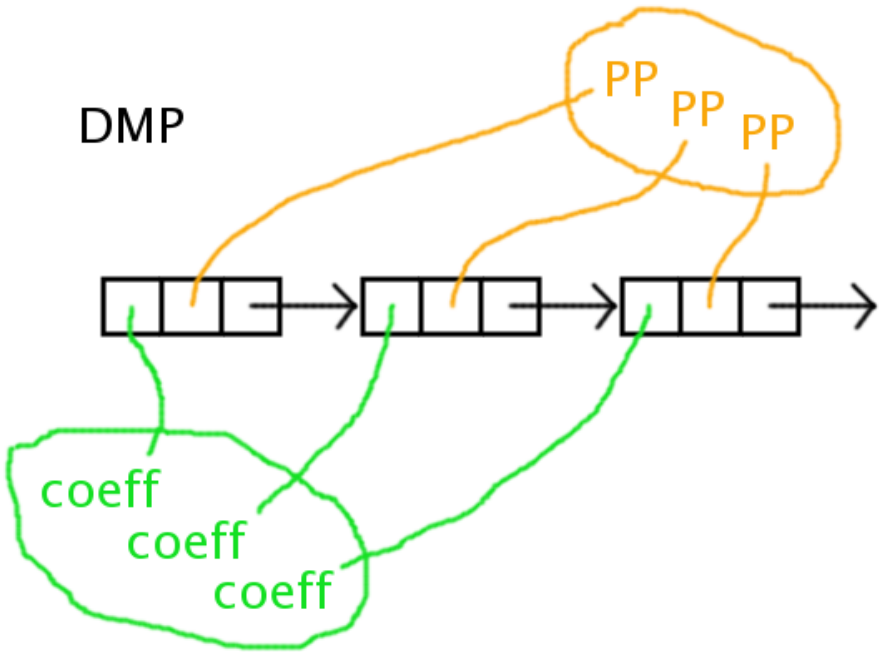
CoCoALib with an analog C++ (STL) function:

push_back	PushBack
empty	IsEmpty

Some files in CoCoALib are called **Tmp...**: usually undocumented code, the operation will become official, but syntax might change.

Polynomials

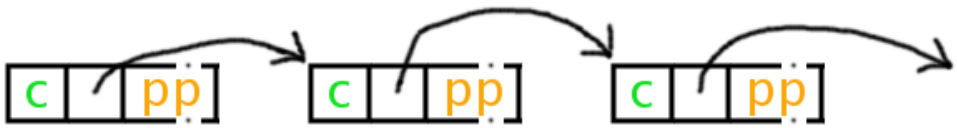
RingDistrMPoly, RingDistrMPolyInlPP, RingDistrMPolyInlFpPP



+ clean, easy to maintain, completely general

– poor locality, slow over F_q

DMPII (in some special cases)



+ good locality, fast

– less clean, harder to maintain

Polynomial iterators

From ex-PolyIterator1.C

```
for (SparsePolyIter i=BeginIter(f); !IsEnded(i); ++i)
{
  cout << "coeff: " << coeff(i)
       << "\t   element of " << owner(coeff(i)) << endl
       << "PP: " << PP(i)
       << "\t   element of " << owner(PP(i)) << endl
       << endl;
}
```

Power products (monomials/terms)

- PPMonoidEv exponent vector
- PPMonoidEvOv exponent vector and order vector
- PPMonoidEvZZ ZZ exponent vector (for very high exponents)
- PPMonoidOv order vector (default for RingDistrMPolyInl(Fp)PP)
- PPMonoidSparse sparse representation

(Run *ex-PPMonoidElem2.C*)

Documentation

- text (t2t), pdf, html: corresponding to the .H/.C files
- doxygen: automatic from the sources and comments (outdated)
- `examples/` directory:
  - focus on a class and give all its functions e.g. `ex-RingElem1.C`
  - "pieces of code" explaining particular functions, e.g. `ex-PolyRing1.C`
  - workarounds for missing or incomplete aspects, e.g. `ex-AlexanderDual.C`

Some Future Plans

- CoCoA-5, new interactive system with improved language & better errors
- self-saturating algorithm for non-homogeneous Gröbner bases
- redesign implementation of ideals (monomial ideals, ideals of points)
- see CoCoALib Task Table for more details

How to join in

Prerequisites

- Some knowledge of basic C++ programming
- Mild familiarity with compilation and `make`
- the GMP library

– Visit CoCoA web page

What to do

- Download CoCoALib from `http://cocoa.dima.unige.it/cocoalib/`
  current version: `CoCoALib-0.9944`
- Configure and compile
  `./configure; make`
- Play and experiment!
  `cd examples; make`

– Compile and run (Run *ex-empty.C*)