New in CoCoA-5.2.4 and CoCoALib-0.99600 for SC-Square

John Abbott[1], Anna M. Bigatti[1], and Elisa Palezzato[2]

[1] Dipartimento di Matematica, Università degli Studi di Genova, Italy
[2] Department of Mathematics, Hokkaido University

Abstract. CoCoALib is a C++ software library offering operations on polynomials, ideals of polynomials, and related objects. The principal developers of CoCoALib are members of the SC^2 project. We give an overview of the latest developments of the library, especially those relating to the project SC^2.
The CoCoA software suite includes also the programmable, interactive system CoCoA-5. Most of the operations in CoCoALib are also accessible via CoCoA-5. The programmability of CoCoA-5 together with its interactivity help in fast prototyping and testing conjectures.

1 Introduction

We briefly recall what is described in [2] and [3]. The CoCoA project dates back to 1987, with the first public release of its interactive system in 1989. The main purpose of the project has always been to offer a convenient software laboratory for studying *Computational Commutative Algebra*, especially ideals of multivariate polynomials (*e.g.* Gröbner bases).

The CoCoA software has a modular design: its "mathematical expertise" resides in the C++ software library [4], while the interactive system [7] uses an interpreter which grants easy access to CoCoALib's capabilities (without requiring any knowledge of C++). All code is free and open source (under licence GPL-3).

We give an overview of the latest developments of the library and of the system (for the summer 2018 release). There are some new aspects of particular interest to the project SC^2. One feature is an implementation in CoCoALib of a new efficient algorithm for computing the *minimal polynomial* ([5, 3]), and its application to many operations on 0-dimensional ideals — a prototype implementation in CoCoA-5 was mentioned in [3]. In particular, in view of applications to Cylindrical Algebraic Decomposition (CAD), we focus on factoring polynomials over algebraic field extensions, and on evaluating good approximations for their roots.

Another feature of interest to SC^2 is the prototype interface for communication between CoCoALib and MathSAT (Section 4).

2 Improving usability of CoCoA for SC²

2.1 Timeout mechanism

A flexible "timeout" mechanism has been added to CoCoALib. It has a simple user interface, and can be used with several function calls to CoCoALib. The exact behaviour of the timeout depends on the specific function: *e.g.* some functions either return the correct and complete answer within the requested time limit, or throw an exception to say that the result could not be computed quickly enough; other functions, which compute approximations to some exact value, return the best approximation which could be obtained within the given time limit.

One purpose of the timeout mechanism is to allow a "speculative" approach to solving: *i.e.* a potentially costly algorithm may be called with a time limit, and in fortunate cases the answer is returned, but in the worst case of a vain attempt only the specified amount of time has been consumed. For example, one may attempt to find all real solutions to a 0-dimensional polynomial system (internally this computes a Gröbner basis); if successful then the result is valuable, but in some unlucky cases the Gröbner basis computation may be unreasonably slow, so we use the timeout mechanism to avoid the "black hole", and must then proceed without knowing what the solutions to the system are — this technique has already been successfully employed inside CoCoALib itself: for example, in testing primality of zero-dimensional ideals (which is called during the factorization of polynomials over algebraic extensions, Section 3.1).

2.2 Towards Real Solving

We have implemented a prototype for `GBasisRealSolve` which removes some non-real components during Gröbner basis computation. The critical internal operation is a quick function for "approximating" the real radical of a polynomial. In this context "approximating" means applying quick heuristics for determining whether a polynomial admits any real solutions; the heuristic may respond `true`, `false` or `uncertain`. Factors which surely have no real roots are removed, whereas those which do have or may have real roots are retained.

The heuristic employs two other ideas mentioned here: real root counting (via Sturm Sequences), and timeout. Even though this heuristic approach is, mathematically speaking, not a satisfactory solution to the general problem, the function `GBasisRealSolve` seems to work well on examples arising from SMT computations (see Section 4).

A mathematically complete solution could go via *Cylindrical Algebraic Decomposition.* Towards this end, we have developed in CoCoA factorisation of polynomials over algebraic extensions, and evaluation of polynomials over real intervals (Section 3).

3 Algebraic extensions

CoCoALib has had for some time the capability to compute with polynomials whose coefficients lie in an algebraic extension. The extension may be specified by any maximal ideal, and does not require that a primitive element be furnished. One important step for CAD is the computation of isolating intervals for the real roots of a polynomial with coefficients in a (real) algebraic extension. CoCoALib is not yet able to do this last operation, but we are developing the various components necessary to reach this goal.

3.1 Factorization over algebraic extensions

In this section we describe an effective method for factorizing univariate polynomials over finite field extensions. This method, based on the computation of primary decompositions of zero-dimensional ideals, is described in the PhD thesis [10] of the third author, and took inspiration from [11] and [9]; this work included a full implementation in CoCoA-5 language. It has now been streamlined and ported into CoCoALib-0.99600 (together with all functions derived from the computation of the minimal polynomial — see [5]).

Example 1. Let $L = \mathbb{F}_2[a]/\mathcal{M} \cong \mathbb{F}_8$ be the base field, where $\mathcal{M} = \langle a^3 + a + 1 \rangle$ is a maximal ideal, and consider $f(x) = x^5 + x + 1 \in L[x]$, the polynomial to be factorized.

```
/**/ FF_2 ::= ZZ/(2);
/**/ use S ::= FF_2[a];
/**/ M := ideal(a^3+a+1);
/**/ L := S/M;
/**/ use Lx ::= L[x];
/**/ f := x^5+x+1;
/**/ factor(f);
record[
  RemainingFactor := (1),
  factors := [x+(a^2+a+1),  x+(a^2+1),  x+(a+1),  x^2+x+(1)],
  multiplicities := [1,  1,  1,  1]
]
```

This is the algorithm implemented in CoCoALib. Let $S = K[a_1, \ldots, a_m]$ be a polynomial ring over a perfect field K, let $\mathcal{M}$ be a maximal ideal in S, let $L = S/\mathcal{M}$, and let $f(x) \in L[x]$ be a univariate polynomial. The factorization of $f(x)$ is obtained by computing the primary decomposition of the ideal $\mathcal{M}+\langle f(x) \rangle$ in the polynomial ring $K[a_1, \ldots, a_m, x]$; for the proof of correctness see [10].

Algorithm for Factorization over Algebraic Extension
Notation: let $S = K[a_1, \ldots, a_m]$, $\mathcal{M}$ a maximal ideal in S, $L = S/\mathcal{M}$
Input $f(x)$ polynomial in $L[x]$

1 create the ring $P = K[a_1, \ldots, a_m, x]$
  let $\phi\colon S \longrightarrow P$, defined by $a_i \mapsto a_i$
  let $\psi\colon P \longrightarrow L[x]$, defined by $a_i \mapsto \bar{a}_i$ and $x \mapsto x$
2 let $\mathcal{M}P = \langle \phi(g) \mid g \in \text{gens}(\mathcal{M}) \rangle \subseteq P$
  let $f_P \in \psi^{-1}(f)$, a representative of f in P
  let $J = \mathcal{M}P + \langle f_P \rangle$ ideal in P
3 compute the primary decomposition of $J \longrightarrow Q_1 \cap \cdots \cap Q_s$
4 compute g_i, the monic generator of $\psi(Q_i)$ in $L[x]$ — note: $L[x]$ is PID
5 compute $h_i = \text{rad}(g_i)$ and let $m_i = \frac{\deg(g_i)}{\deg(h_i)}$

Output $\text{LC}(f) \cdot \prod_{j=1}^{s} h_i^{m_i}$, the factorization of f in $L[x]$

Example 2. The internal computation for Example 1, following the algorithm above, actually performs these steps:

```
/**/ use P ::= FF_2[a, x];
/**/ phi := PolyAlgebraHom(S, P, [a]);
/**/ psi := PolyRingHom(P, Lx, CanonicalHom(FF_2, L),
                         [RingElem(Lx,"a"), RingElem(Lx,"x")]);
/**/ J := ideal(a^3+a+1) + ideal(x^5+x+1);
/**/ PrDec := PrimaryDecomposition(J);   PrDec;
[ ideal(a^3 +a +1,  x^5 +x +1,  x^2 +x +1),
  ideal(x^3 +x^2 +1,  a^3 +a +1,  a^2*x +a*x^2 +a^2 +a),
  ideal(x^3 +x^2 +1,  a^2 +a*x +x^2 +a,  a*x^2 +x +1),
  ideal(x^3 +x^2 +1,  a^2 +a*x +x^2 +a,  a*x^2 +x)          ]
/**/ [ ReducedGBasis(ideal(apply(psi,gens(Q)))) | Q in PrDec ];
[[x^2 +x +(1)],  [x +(a^2+a+1)],  [x +(a^2+1)],  [x +(a+1)]]
```

and these are indeed the (square-free) factors of f.

Next we see a call to `factor` in an algebraic extension which is given by a non-principal, maximal ideal, $L = \mathbb{Q}[a, b]/\langle a^2 - 3,\ b^2 - ab - 4 \rangle$,

```
/**/ use S ::= QQ[a, b];
/**/ M := ideal(a^2-3, b^2-a*b-4);
/**/ IsMaximal(M); // check that M is a maximal ideal
true
/**/ L := S/M;
/**/ use Lx ::= L[x];
/**/ factor(x^6 +(b^2)*x^4 +(-55*b^2+80)*x^2 +(315*b^2-528));
record[
  RemainingFactor := (1),
  factors := [x^2 +(3*b^2),  x +(-b),  x +(b)],
  multiplicities := [1,  2,  2]
]
```

3.2 Counting real roots

We have implemented a function for computing a primitive *Sturm sequence* of a given (univariate) polynomial with rational coefficients. Sturm sequences enable

one to compute easily the number of real roots a given (univariate) polynomial has in a given real interval; in particular, they can be used to tell whether a given (univariate) polynomial has any real roots. CoCoA also offers a function to count the number of real roots:

```
/**/ W := product([x-k | k in 1..20]);//Wilkinson's polynomial
/**/ W2 := W + (1/7402570310)*x^19;
/**/ NumRealRoots(W2);
18
/**/ W3 := W + (1/7402570311)*x^19;
/**/ NumRealRoots(W3);
20
```

The interactive CoCoA-5 system offers a suite of interpreted functions for computing tight isolating intervals for real roots. This code is not yet available in CoCoALib, but we are planning to port it (though this lengthy task will likely be completed after the end of this first SC^2 project). While a list of isolating intervals gives more explicit information than a Sturm sequence, it is also generally more costly to compute.

3.3 Bounds on roots of polynomials

CoCoA now also offers a collection of functions for obtaining a good upper bound for the absolute value of every *complex root* of a given (univariate) polynomial. The main function strikes an automatic balance between speed of computation and tightness of the computed bound; an optional second argument lets the caller choose a different balance. A root bound gives less information than a Sturm sequence but can be computed faster, and may sometimes give enough information to decide unsolvability. A root bound can be computed for any non-constant (univariate) polynomial, but gives no indication whether any of the roots is real.

```
/**/ H := HermitePoly(50, x); // largest real root is 9.18
/**/ RootBound(f);      // fair compromise for speed/accuracy
295/32                  // 9.22
/**/ RootBound(f,0);    // fastest, 0 iterations
237/8                   // 29.625  (quite loose)
/**/ RootBound(f,1);    // still fast, 1 iteration
115/8                   // 14.375  (better than 0 iters)
```

3.4 Interval arithmetic, and range of a polynomial

We have recently added to CoCoALib a prototype suite of functions for "interval arithmetic" on closed real intervals with rational endpoints. The suite includes a more advanced function for evaluating a given (univariate) polynomial with rational coefficients (or interval coefficients) over a given interval. The result is another interval, comprising effectively a lower bound for the minimum and an upper bound for the maximum for the polynomial over the given interval. In

general, the result contains strictly the true interval of reachable values: in fact, the true range interval may have irrational end points; *e.g.* if $f = x^3 - x$ then its range over the interval $[-1, 1]$ has both end points being irrational, namely $[-\frac{2\sqrt{3}}{9}, \frac{2\sqrt{3}}{9}]$. Consequently, when using intervals with rational end points only an approximation to the correct answer can be produced.

We are still studying the compromise between speed of computation and tightness of the resulting interval. Naturally, it can help answer some questions of solvability where both the variable and the value of the polynomial are limited to finite intervals. One future aim is to use this suite to allow isolation of real roots of polynomials whose coefficients are real algebraic numbers (each represented as a small isolating interval together with the minimal polynomial).

As an example we can compute the range of values of the 10-th Chebyshev polynomial evaluated on the interval $I = [-1, 1]$. From the theory we know that the true answer is the interval $[-1, 1]$; our current prototype implementation produces a fair approximation, being the wider interval from -1.15 to 1.12. The Chebyshev polynomials are an interesting test case because they have many local maximums and minimums, which become global maximums and minimums when we restrict the domain to the interval I. We do not give an explicit example in CoCoA-5 since the user interface is not yet settled.

4 Interface MatSAT↔CoCoA-5

In [3] we described the first steps in the communication between CoCoALib and MathSAT. Some developments followed, in close collaboration with Alberto Griggio.

The construction in CoCoALib of polynomial rings with user defined orderings (such as elimination orderings) has been reorganized so that it makes fewer repeated checks on the admissibility of the term-ordering. This is not a problem in the usual context of Commutative Algebra, where the cost of the computation of a Gröbner basis exceeds by far the time spent in the preliminary construction of the polynomial ring. But the first collection of examples arising from MathSAT computations, highlighted that this is indeed an issue when we deal with many indeterminates and relatively simple Gröbner basis computations.

The communication between CoCoALib and MathSAT has been improved, so the overhead for the data conversions is now negligible.

Indeed, thanks to this collaboration both CoCoALib and MathSAT speeded up their code in some parts which had never been highlighted during their respectively *normal* usage.

Using this new interface, we have built a first collection of examples of systems of polynomial equalities and inequalities generated from MathSAT computations, for experimenting with CoCoA. Since CoCoA natively handles only equalities, we applied an input manipulation which converts MathSAT inequalities into CoCoA equalities by adding squares of *slack variables*, with further increase of the number of indeterminates. Then we tested the function `GBasisRealSolve`

(Section 2.2), using a specific weighted graduation and term-ordering, and observed it works very effectively on these examples, detecting *UNSAT* in remarkably few steps.

We wrote a prototype CoCoA-5 package using `MSatLinSolve`, the first MathSAT function in CoCoA-5 (see [3]) for the computation of Gröbner fan as an alternative to the call to `GFanRelativeInteriorPoint` (part of the communication between CoCoALib and Gfanlib [8]). This is an intriguing application; it is not (yet) competitive with the original code for two main reasons, partly because the non-trivial code preparing the input for `MSatLinSolve` is written in the interpreted language of CoCoA-5, but especially because the adaptation of the original CoCoA code was not designed to exploit incrementality, which is the strong point of MathSAT.

References

1. J. Abbott, *Fault-Tolerant Modular Reconstruction of Rational Numbers*, J. Symb. Comp. 80P3, pp. 707–718, 2017.
2. J. Abbott, A.M. Bigatti, *CoCoA and CoCoALib: Fast prototyping and flexible C++ library for Computations in Commutative Algebra* Proc. 1st Workshop on Satisfiability Checking and Symbolic Computation, SC-Square 2016, CEUR Workshop Proceedings 1804, pp. 1–3, 2016.
3. J. Abbott, A.M. Bigatti, *CoCoA and CoCoALib: Fast prototyping and flexible C++ library for Computations in Commutative Algebra* Proc. 2nd Workshop on Satisfiability Checking and Symbolic Computation, SC-Square 2017, CEUR Workshop Proceedings 1974, pp. 1–6, 2017.
4. J. Abbott, A.M. Bigatti, *CoCoALib: a C++ library for doing Computations in Commutative Algebra* Available at `http://cocoa.dima.unige.it/cocoalib`
5. J. Abbott, A. Bigatti, E. Palezzato, L. Robbiano, *Computing and Using Minimal Polynomials*, `arXiv:1704.03680`, 2017.
6. J. Abbott, A. Bigatti, L. Robbiano, *Ideals modulo p*, In preparation.
7. J. Abbott, A.M. Bigatti, L. Robbiano *CoCoA: a system for doing Computations in Commutative Algebra* Available at `http://cocoa.dima.unige.it/`
8. A.N. Jensen, *Gfan, a software system for Gröbner fans and tropical varieties.* Available from `http://home.math.au.dk/jensen/software/gfan/gfan.html`
9. M. Kreuzer and L. Robbiano, *Computational Linear and Commutative Algebra*, Springer, Heidelberg 2016.
10. E. Palezzato, *Minimal Polynomials, Sectional Matrices, and Applications*; PhD. thesis, Università degli Studi di Genova, 2017.
11. Sun Y. and Wang D., *Efficient algorithm for factoring polynomials over algebraic extension fields*, Sci. China Math., 56, 1155-1168, 2013.