

Giac and GeoGebra – Improved Gröbner Basis Computations

Zoltán Kovács¹(✉) and Bernard Parisse²

¹ Johannes Kepler University, Altenberger Strasse 54, 4040 Linz, Austria
zoltan@geogebra.org

² Institut Fourier, UMR 5582 du CNRS, Université de Grenoble, 100 Rue des Maths,
BP 53, 38041 Grenoble Cedex 9, France
bernard.parisser@ujf-grenoble.fr

Abstract. GeoGebra is open source mathematics education software being used in thousands of schools worldwide. It already supports equation system solving, locus equation computation and automatic geometry theorem proving by using an embedded or outsourced CAS. GeoGebra recently changed its embedded CAS from Reduce to Giac because it fits better into the educational use. Also careful benchmarking of open source Gröbner basis implementations showed that Giac is fast in algebraic computations, too, therefore it allows heavy Gröbner basis calculations even in a web browser via Javascript.

Gröbner basis on \mathbb{Q} for revlex ordering implementation in Giac is a modular algorithm (E. Arnold). Each $\mathbb{Z}/p\mathbb{Z}$ computation is done via the Buchberger algorithm using F4 linear algebra technics and “remake” speedups, they might be run in parallel for large examples. The output can be probabilistic or certified (which is much slower). Experimentation shows that the probabilistic version is faster than other open-source implementations, and about 3 times slower than the Magma implementation on one processor, it also requires less memory for big examples like Cyclic9.

Keywords: Gröbner basis · Computer algebra · Computer aided mathematics education

1 Introduction: Heavy Computations in the Classroom

Mathematics education has always been influenced by culture and traditions, nevertheless development of technology also played an important role in changing the approach of the teacher and the subject of teaching. The availability of personal computers and the computer algebra systems (CAS) being widespread made possible to verify the results of manual solving of an equation, what is more, to solve equations automatically and concentrate on higher level problems in the classroom.

Solving an equation system in the high school is one of the most natural mathematical problems, required not only by pure mathematics but physics and

chemistry as well. Despite of its importance the way of solving an equation system is not trivial. In general, there is no algorithmic technique known which covers all possible equation systems and returns all solutions of a problem in finite time. As Wikipedia explains,

In general, given a class of equations, there may be no systematic method (algorithm) that is guaranteed to work. This may be due to a lack of mathematical knowledge; some problems were only solved after centuries of effort. But this also reflects that, in general, no such method can exist: some problems are known to be unsolvable by an algorithm, such as Hilbert’s tenth problem, which was proved unsolvable in 1970 [20].

Even after some restrictions (for example assuming that the equations are algebraic and the solutions are real numbers) there is no guarantee that the system will be solvable quickly enough, however there are efficient methods already which can compute a large enough set of problems of “typical uses”.

On the other hand, classroom use of modern equation solver algorithms (in the background, i.e. invisibly for the students) cannot be restricted to direct equation solving only. There are other fields of mathematics which seem distant or unrelated with computer algebra, but still use heavy algebraic computations. With no doubt two such fields in analytical geometry are automated theorem proving (for computing proofs for Euclidean theorems in elementary geometry, see Fig. 1) and locus computation (for example to introduce the notion of parabola analytically [14]).

In most classroom situations there is no importance in the applied algorithm when an algebraic equation system must be solved in the background. Some advanced problems which are normally present in the high school curriculum, however, may lead to slow computations if the applied algorithm is not fast enough.

In this paper first we show some possible classroom situations from the present challenges of mathematics education by utilizing the dynamic geometry software GeoGebra. In Sect. 2 we demonstrate the bottleneck of GeoGebra’s formerly used CAS (Reduce [9]) in its web based version, and show an alternative CAS (Giac [18]) for the same modern classroom. Then we focus on benchmarking Giac’s and other CAS’s especially in computing solutions of algebraic equation systems, i.e. the Gröbner basis of a set of polynomials. In Sect. 3 we provide the main concepts of Giac’s Gröbner basis algorithm with detailed benchmarks.

2 Computer Algebra in the Classroom

The introduction of smartphones and tablets with broadband Internet connection allows students to access educational materials from almost everywhere. Ancsin et al. [1] argues that today’s classroom computers are preferably no longer workstation PCs but tablets. Thus one of the most important question for a software developer working for mathematics education is “will my program

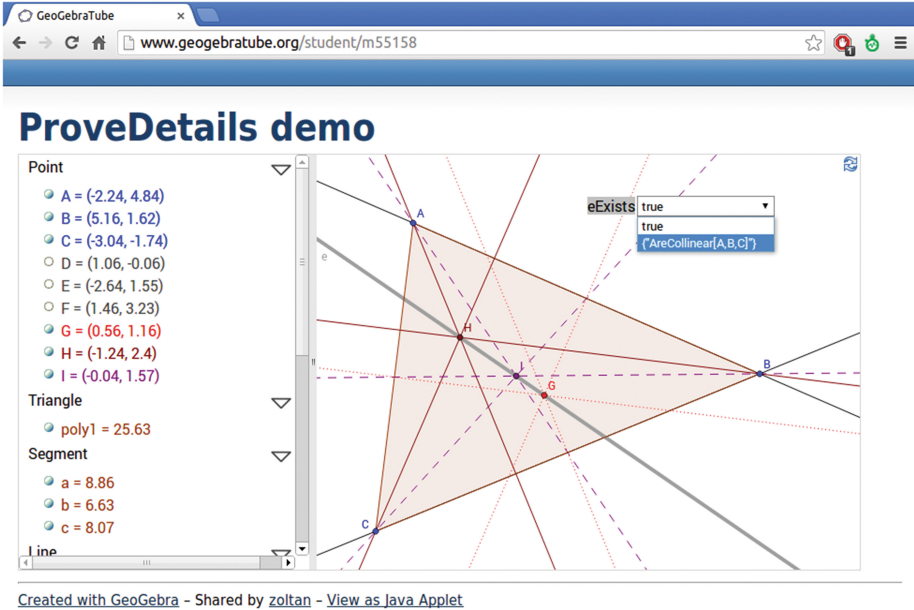


Fig. 1. Giac computes sufficient condition for the Euler’s line theorem in JavaScript. Here the elimination ideal of $2v_7 - v_3, 2v_8 - v_4, 2v_9 - v_5 - v_3, 2v_{10} - v_6 - v_4, -v_{11}v_{10} + v_{12}v_9, -v_{11}v_8 + v_{12}v_7 + v_{11}v_6 - v_7v_6 - v_{12}v_5 + v_8v_5, -v_{14} - v_5 + v_3, -v_{13} + v_6 - v_4, -v_{16} + v_6 + v_3, -v_{15} + v_5 - v_4, v_{17}v_{14} - v_{18}v_{13}, v_{17}v_{16} - v_{18}v_{15} - v_{17}v_6 + v_{15}v_6 + v_{18}v_5 - v_{16}v_5, 2v_{19} - v_5 - v_3, 2v_{20} - v_6 - v_4, v_{22} - v_{20} - v_{19} + v_3, v_{21} + v_{20} - v_{19} - v_4, 2v_{23} - v_3, 2v_{24} - v_4, v_{26} - v_{24} - v_{23} + v_3, v_{25} + v_{24} - v_{23} - v_4, -v_{27}v_{22} + v_{28}v_{21} + v_{27}v_{20} - v_{21}v_{20} - v_{28}v_{19} + v_{22}v_{19}, -v_{27}v_{26} + v_{28}v_{25} + v_{27}v_{24} - v_{25}v_{24} - v_{28}v_{23} + v_{26}v_{23}, -1 + v_{29}v_{27}v_{18} - v_{29}v_{28}v_{17} - v_{29}v_{27}v_{12} + v_{29}v_{17}v_{12} + v_{29}v_{28}v_{11} - v_{29}v_{18}v_{11}$ is computed with respect to revlex ordering for variables $v_7, v_8, v_9, \dots, v_{29}$. The result ideal contains $v_3v_6 - v_4v_5$ which yields the geometrical meaning “if triangle ABC is non-degenerate, then its orthocenter H , centroid I and circumcenter G are collinear”.

work on a tablet”? Technically speaking, modern developments should move to the direction of the HTML5 standard with JavaScript (JS) empowered.

Online access of web server based computer algebra systems became very popular in the academic world during the last years, including many students and teachers of a number of universities world-wide. For example, *Sage*¹ has already been famous not only for being freely available to download, but for its free demonstration server <http://cloud.sagemath.com>. Similar approaches are the *SymPy Live/Gamma*² projects and the *IPython Notebook*³. On one hand, these systems are free of charge and thus they can be well used in education, and they are empowered by HTML5 and JS on the client side. On the other hand,

¹ <http://sagemath.org>.
² <http://live.sympy.org>, <http://www.sympygamma.com>.
³ <http://ipython.org/notebook.html>.

when no or only slow Internet connection is available, none of these systems can be used conveniently because the server side computations are not available any longer. This is why it seems to be a more fruitful approach to develop an *offline* system which can be run locally on the user’s machine inside a web browser, especially in those classrooms where no Internet connection is permitted.

GeoGebra developers, reported in [16], started to focus on implementing a full featured offline CAS using the HTML5 technology. The first visible result in May 2012 was the embedded system GGBReduce which offered many modules of the Reduce CAS, using the Google Web Toolkit for compiling the JList Lisp implementation into JS. This work is a official part of Reduce under the name JSLisp now [12].

The first tests back in May 2011 were very promising: the <http://www.geogebra.org/mpreduce/mpreduce.html> web page with the base system loaded below 1 s and used 1.7 MB of JS code. Unfortunately, after adding some extra modules and the Lisp heap, the initialization time of the Reduce system increased drastically: see http://dev.geogebra.org/qa/?f=_CAS_mixture.ggb&c=w-42-head for an example of GeoGebraWeb 4.2 (2.2 MB JS), the notification message “CAS initializing” disappears only after 10 s or even more. By contrast, http://dev.geogebra.org/qa/?f=_CAS_mixture.ggb&c=w-44-head loads below 5 s, using the Giac CAS in GeoGebraWeb 4.4 (7.5 MB JS).

Test case http://dev.geogebra.org/qa/?f=_CAS_commands.ggb&... shows some typical classroom computations related to analyze a rational function as an exercise. It includes factorizing a polynomial, turning a rational function into partial fractions, computing the limit of a rational function at infinity, and computing the asymptotes. Also conversion of symbolic to numeric, computation of derivative and a solution of an equation in one variable are included. Finally, GeoGebraWeb 4.4 displays the graph of the function and its derivative. The entire process completes in 7 s. Each update takes 5 s.

These benchmarks were collected, however, on a modern PC. On low cost machines the statistics will be worse, expecting further work on speedup the computations preferably by using native code in the browser if possible.

2.1 JavaScript: The Assembly Language for the Web

Embedding a third party CAS usually requires a remarkable work in both the main software and the embedded part. Figure 2 shows former efforts to include a CAS in GeoGebra. JSCL, Jasymlca [4], Jama [10] and Yacas/Mathpiper [13] were used as native Java software, but Reduce is written in Lisp and Giac in C++. (Not shown in the figure, but also Maxima was planned as an extra CAS shipped with GeoGebra separately—this plan was finally cancelled.)

The main criteria for using a CAS was to be able to run in both the “desktop” and “web” environments. The desktop environment, technically a Java Virtual Machine, has used to be the primary user interface for many years since the very beginning of the GeoGebra project, including platforms Microsoft Windows, Apple’s Mac OS X, and Linux. The web environment is the new direction,

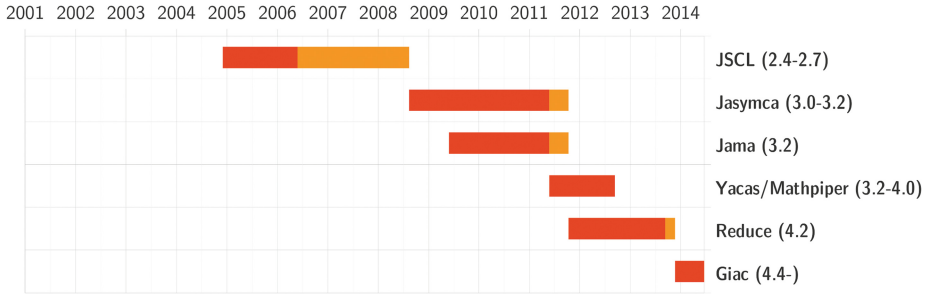


Fig. 2. Computer algebra systems used in GeoGebra from version 2.4 to 4.4. Orange bars show that the corresponding version of GeoGebra was no longer developed by the programmers, but still used by the community (Color figure online).

being capable of supporting platforms Windows 8, iPad, Android, and Google’s Chromebook.

Our approach was to support both environments with the same codebase, i.e. to make it possible to use the same source code for all platforms. This has been succeeded by using the tools explained in Fig. 3. The speedup in the initialization time is obvious: Giac is much faster in its startup, even if there is a slowness factor between 1 and 10 compared to the Java Native Interface (JNI) version, depending on the type of the computation.

GeoGebra since version 4.2 not only supports typical CAS operations but dynamic geometry computations as well. An example is the **LocusEquation** command which computes the equation of the locus (if its construction steps can be described algebraically). The forthcoming version 5.0 will support **Envelope** equations and automated geometry proofs in elementary geometry by using the **Prove** command. It was essential to make benchmarks to test the underlying algebra commands in Giac before really using them officially in 4.4.

“JavaScript is Assembly Language for the Web”, states Scott Hanselman from Microsoft, citing Erik Meijer, former head of Cloud Programmability Team at Microsoft [8]. The reasoning is as follows:

- JavaScript is ubiquitous.
- It’s fast and getting faster.
- Javascript is as low-level as a web programming language goes.
- You can craft it manually or you can target it by compiling from another language.

On the other hand, despite being universal and fast, algebraic computations programmed in a non-scripting (i.e. compiled) language (e.g. C or Java) are still much faster than it is expectable for being run in a web browser using the normal browser standards. After the JavaScript engine race 2008–2011 [22], there is a second front of bleeding edge research to develop another standard language of the web (Google’s Dart [23], for example), and a renewed focus on C and C++ to compile them into browser independent or native bytecode

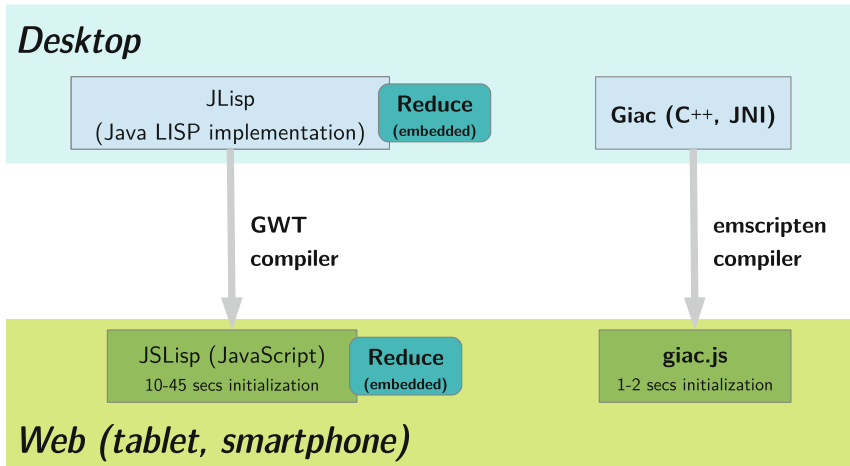


Fig. 3. Embedded computer algebra systems Reduce and Giac in GeoGebra 4.2 and 4.4. Reduce itself is embedded into JLisp and JSLisp.

(e.g. Google’s Portable and Native Client [21])⁴. Despite these new experimental ways, JavaScript is still the de facto standard of portability and speed for modern computers, and probably the best approach to generate as fast JavaScript code for computer algebra algorithms as possible.⁵

To have an exact speed comparison of JNI and JS versions of Giac a modern headless scriptable browser will be helpful. The new version of *PhantomJS*⁶ has already technical support to run Giac especially for benchmarking purposes: its stable version is expected to be publicly available soon.

2.2 Benchmarks from Automated Theorem Proving

The GeoGebra Team already developed a benchmarking system for testing various external computer algebra systems with different stress cases. In this

⁴ Giac has already been successfully compiled into .pexe and .nexe applications at <http://ggbl.idm.jku.at/~kovzol/data/giac>. The native executables for 32/64 bits Intel and ARM architecture binaries are between 9.7 and 11.9 MB, the portable executable is 6.3 MB, however, the .pexe → .nexe compilation takes too long, at least 1 min on a recent hardware. The runtime speed is comparable with the JNI version, i.e. much faster than the JavaScript version.

⁵ Axel Rauschmayer, author of the forthcoming O’Reilly book *Speaking JavaScript*, predicts JavaScript to run near-native performance in 2014 (see <http://www.2ality.com/2014/01/web-platform-2014.html> for details). The speed is currently about 70 % of compiled C++ code by using `asm.js`. This will, however, doubtfully speed up the JS port of Giac like in a native client since it heavily uses the anonymous union of C/C++ to pack data (unsupported in JS).

⁶ <https://github.com/ariya/phantomjs/wiki/PhantomJS-2> contains a step-by-step guide to compile the development version of PhantomJS version 2.

subsection we simply show the result of this benchmarking. In the next section other tests will be shown between systems designed much more for algebraic computations.

In this subsection we run the tests on *open source* candidates. GeoGebra is an open source application due to its educational use: schools and universities may prefer using software free of charge than paying for licenses.

The test cases are chosen from a set of simple theorems in elementary geometry. All tests are Gröbner basis computations in a polynomial ring over multiple variables. The equation systems (i.e. the ideals to compute the Gröbner basis for) can be checked in details at [15], the final summary (generated with default settings in Giac as of September 2013) is shown in Table 1.

The conclusion of the statistics was that Giac would be comparable with the best open source algebraic computation software, Singular. The first impression of its speed was that it is good competitor of Reduce, thus a very good candidate to be a long term basis for all CAS computations in GeoGebra. (Later Giac was extended with an even faster algorithm for computing Gröbner basis, as described in Sect. 3.)

3 Gröbner Basis Algorithm in Giac⁷

Starting with version 1.1.0-26, Giac [18] has a competitive implementation of Gröbner basis for reverse lexicographic ordering, that will be described in this section.

3.1 Sketch of E. Arnold Modular Algorithm

Let f_1, \dots, f_m be polynomials in $\mathbb{Q}[x_1, \dots, x_n]$, $I = \langle f_1, \dots, f_m \rangle$ be the ideal generated by f_1, \dots, f_m . Without loss of generality, we may assume that the f_i have coefficients in \mathbb{Z} by multiplying by the least common multiple of the denominators of the coefficients of f_i . We may also assume that the f_i are primitive by dividing by their content.

Let $<$ be a total monomial ordering (for example `revlex` the total degree reverse lexicographic ordering). We want to compute the Gröbner basis G of I over \mathbb{Q} (and more precisely the inter-reduced Gröbner basis, sorted with respect to $<$). Now consider the ideal I_p generated by the same f_i but with coefficients in $\mathbb{Z}/p\mathbb{Z}$ for a prime p . Let G_p be the Gröbner basis of I_p (also assumed to be inter-reduced, sorted with respect to $<$, and with all leading coefficients equal to 1).

Assume we compute G by the Buchberger algorithm [3] with Gebauer and Möller criterion [7], and we reduce in \mathbb{Z} (by multiplying the s-poly to be reduced by appropriate leading coefficients), if no leading coefficient in the polynomials are divisible by p , we will get by the same process but computing modulo p the G_p Gröbner basis. Therefore the computation can be done in parallel in

⁷ The content of this section is released under the Public Domain.

Table 1. Outputs of Gröbner basis benchmarking in seconds on an Intel Xeon CPU E3-1220 V2 @ 3.10 GHz running Ubuntu Linux 11.10 in VirtualBox 4.2.10 on an Ubuntu 12.04.1 host. Timeout is 60s, timed out tests are shown with empty cells. Average* shows the average by computing 60s computation time for tests being timed out. Singular solved the first four tests below 0.01—the benchmarking system was unable to measure timing under this precision. The CoCoA column shows the results of Giac via CoCoAlib.

Test	Maxima	JAS 2	Reduce	Singular	CoCoA	Giac
Thales	0.2	0.46	0.11	0.00	0.08	0.03
Heights	0.29	0.51	0.11	0.00	0.29	0.03
Medians	0.4	0.65	0.12	0.00	0.14	0.09
Bisectors	0.42	0.5	0.1	0.00	0.09	0.01
Euler’s line		1.66	0.2	0.01	0.14	0.01
Nine points circle	1.19	1.5	0.11	0.01	0.13	0.01
Angle bisector	36.08	1.74	0.75	0.01	0.31	0.04
Simson’s line						
Pappus		3.37		0.5	9.28	4.9
Simson (reduced)		5.77	6.07	0.07	0.87	0.15
Pappus (reduced)		2.33	2.18	0.02	0.34	0.4
<i>Average</i>	6.43	1.85	1.08	0.06	1.17	0.57
<i>Average*</i>	56.43	7.85	14.41	6.06	7.17	6.56

\mathbb{Z} and in $\mathbb{Z}/p\mathbb{Z}$ except for a finite set of *unlucky* primes (since the number of intermediate polynomials generated in the algorithm is finite). If we are choosing our primes sufficiently large (e.g. about 30 bits), the probability to fall on an unlucky prime is very small (less than the number of generated polynomials divided by about 2^{30} , even for really large examples like Cyclic9 where there are a few 10^4 polynomials involved, it would be about $1e-5$).

The Chinese remaindering modular algorithm works as follows: compute G_p for several primes, for all primes that have the same leading monomials in G_p , reconstruct $G_{\prod p_j}$ by Chinese remaindering, then reconstruct a candidate Gröbner basis G_c in \mathbb{Q} by rational (Farey) reconstruction. Once it stabilizes, do the checking step described below, and return G_c on success.

Checking steps: check that the original f_i polynomials reduce to 0 with respect to G_c (fast check) and check that G_c is a Gröbner basis (slow check).

Theorem 1 (Arnold). *If the checking steps succeed, then G_c is the Gröbner basis of I .*

This is a consequence of ideal inclusions (first check) and dimensions (second check), for a complete proof, see [2]. The proof does not require that we reconstruct from Gröbner basis for all primes p , it is sufficient to have one Gröbner basis for one of the primes. This can be used to speedup computation like in F4remake (Joux-Vitse, see [11]).

3.2 Computation Modulo a Prime

The Buchberger algorithm with F4[5,6]-like linear algebra is implemented modulo primes smaller than 2^{31} using total degree as selection criterion for critical pairs.

1. Initialize the basis to the empty list, and a list of critical pairs to empty.
2. Add one by one all the f_i to the basis and update the list of critical pairs with Gebauer and Möller criterion, by calling the *gbasis update procedure* (described below at step 9).
3. Begin of a new iteration:
All pairs of minimal total degree are collected to be reduced simultaneously, they are removed from the list of critical pairs.
4. The symbolic preprocessing step begins by creating a list of monomials, gluing together all monomials of the corresponding s-polys (this is done with a heap data structure).
5. The list of monomials is “reduced” by division with respect to the current basis, using heap division (like Monagan-Pearce [17]) without taking care of the real value of coefficients. This gives a list of all possible remainder monomials and a list of all possible quotient monomials and a list of all quotient times corresponding basis element monomial products. This last list together with the remainder monomial list is the list of all possible monomials that may be generated reducing the list of critical pairs of maximal total degree, it is ordered with respect to $<$. We record these lists for further prime runs (speeds up step 4 and 5) during the first prime computation.
6. The list of quotient monomials is multiplied by the corresponding elements of the current basis, this time doing the coefficient arithmetic. The result is recorded in a sparse matrix, each row has a pointer to a list of coefficients (the list of coefficients is in general shared by many rows, the rows have the same reductor with a different monomial shift), and a list of monomial indices (where the index is relative to the ordered list of possible monomials). We sort the matrix by decreasing order of leading monomial.
7. Each s-polynomial is written as a dense vector with respect to the list of all possible monomials, and reduced with respect to the sparse matrix, by decreasing order with respect to $<$. (To avoid reducing modulo p each time, we are using a dense vector of 128 bits integers on 64 bits architectures, and we reduce mod p only at the end of the reduction. If we work on 24 bit signed integers, we can use a dense vector of 63 bits signed integer and reduce the vector if the number of rows is greater than 2^{15}).
8. Then inter-reduction happens on all the dense vectors representing the reduced s-polynomials, this is dense row reduction to echelon form (0 columns are removed first). Care must be taken at this step to keep row ordering for further prime runs.
9. *gbasis update procedure*:
We record zero reducing pairs during the first prime iteration, this information will be used during later iterations with other primes to avoid computing and reducing useless critical pairs (if a pair does not reduce to 0 on \mathbb{Q} ,

it has in general a large number of monomials therefore the probability that it reduces to 0 on the first prime run is very small). Each non zero row will bring a new entry in the current basis. New critical pairs are created with this new entry (discarding useless pairs by applying Gebauer-Möller criterion). An old entry in the basis may be removed if its leading monomial has all partial degrees greater or equal to the leading monomial corresponding degree of the new entry. Old entries may also be reduced with respect to the new entries at this step or at the end of the main loop.

10. If there are new critical pairs remaining start a new iteration at step 3. Otherwise the current basis is the Gröbner basis modulo p .

3.3 Probabilistic and Deterministic Check, Benchmarks

We first perform the fast check that the original f_i polynomials reduce to 0 modulo G_c . Then the user has a choice between a probabilistic fast check (useful for conjectures) and a deterministic slower certification for a computer assisted proof.

Probabilistic checking algorithm: instead of checking that s-polys of critical pairs of G_c reduce to 0, we will check that the s-polys reduce to 0 modulo several primes that do not divide the leading coefficients of G_c and stop as soon as the inverse of the product of these primes is less than a fixed $\varepsilon > 0$ (the check is done only after the reconstructed basis stabilizes, with our examples the first check was always successful).

Deterministic checking algorithm: check that all s-polys reduce to 0 over \mathbb{Q} . The fastest way to check seems to make the reduction using integer computations. We have also tried reconstruction of the quotients over $\mathbb{Z}/p\mathbb{Z}$ for sufficiently many primes: once the reconstructed quotients stabilize, we can check the 0-reduction identity on \mathbb{Z} , and this can be done without computing the products quotients by elements of G_c if we have enough primes (with appropriate bounds on the coefficients of G_c and the lcm of the denominators of the reconstructed quotients).

Benchmarks Comparison of Giac (1.1.0-26) with Singular 3.1 (from Sage 5.10) on Mac OS X.6, Dual Core i5 2.3 Ghz, RAM 2 × 2Go:

- The benchmarks are the classical Cyclic and Katsura benchmarks, and a more random example, described in the Giac syntax below:

```
alea6 := [5*x^2*t+37*y*t*u+32*y*t*v+21*t*v+55*u*v,
39*x*y*v+23*y^2*u+57*y*z*u+56*y*u^2+10*z^2+52*t*u*v,
33*x^2*t+51*x^2+42*x*t*v+51*y^2*u+32*y*t^2+v^3,
44*x*t^2+42*y*t+47*y*u^2+12*z*t+2*z*u*v+43*t*u^2,
49*x^2*z+11*x*y*z+39*x*t*u+44*x*t*u+54*x*t+45*y^2*u,
48*x*z*t+2*z^2*t+59*z^2*v+17*z+36*t^3+45*u];
l:= [x,y,z,t,u,v];
```

```

p1:=prevprime(2^24); p2:=prevprime(2^29);
time(G1:=gbasis(alea6 % p1,1,revlex));
time(G2:=gbasis(alea6 % p2,1,revlex));
threads:=2; // <= to the number of CPU
// debug_infolevel(1); // uncomment for intermed. steps
proba_epsilon:=1e-7; // probabilistic algorithm.
time(H0:=gbasis(alea6,indets(cyclic5),revlex));
proba_epsilon:=0; // deterministic
time(H1:=gbasis(alea6,indets(cyclic5),revlex));

```

- Mod timings were computed modulo `nextprime(2^24)` and modulo 107374-1827 (`nextprime(2^30)`).
- Probabilistic check on \mathbb{Q} depends linearly on log of precision, two timings are reported, one with error probability less than $1e-7$, and the second one for $1e-16$.
- Check on \mathbb{Q} in Giac can be done with integer or modular computations hence two times are reported.

Table 2. Benchmarks between Giac and Singular. » means timeout (3/4 h or more) or memory exhausted (Katsura12 modular $1e-16$ check with Giac) or test not done because it would obviously timeout (e.g. Cyclic8 or 9 on \mathbb{Q} with Singular).

	Giac mod p 24, 31 bits	Giac run2	Singular mod p	Giac \mathbb{Q} prob. $1e-7, 1e-16$	Giac \mathbb{Q} certified	Singular \mathbb{Q}
Cyclic7	0.5, 0.58	0.1	2.0	3.5, 4.2	20, 29.3	»
Cyclic8	7.2, 8.9	1.8	52.5	103, 107	258, 679	»
Cyclic9	633, 1340	200	?	1 day	»	»
Kat8	0.063, 0.074	0.009	0.2	0.33, 0.53	6.55, 4.35	4.9
Kat9	0.29, 0.39	0.05	1.37	2.1, 4.2	54, 36	41
Kat10	1.53, 2.27	0.3	11.65	14, 33.9	441, 335	480
Kat11	10.4, 13.8	2.8	86.8	170, 361	4610, 3530	6035
Kat12	76, 103	27	885	1950, 4000	25000, »	»
Alea6	0.83, 1.08	.26	4.18	187, 194	718	>1h

This leads to the following observations (see Table 2):

- Computation modulo p for 24 to 31 bits is faster than Singular, but seems also faster than Magma (and Maple). For smaller primes, Magma is 2 to 3 times faster.
- The probabilistic algorithm on \mathbb{Q} is much faster than Singular on these examples (this probably means that Singular does not implement a modular algorithm). Compared to Maple 16, it is reported to be faster for Katsura10, and as fast for Cyclic8. Compared to Magma, it is about 3 to 4 times slower.
- If [19] is up to date (except about Giac), Giac is the third software and first open-source software to solve Cyclic9 on \mathbb{Q} (the link is rather old, but we

believe it is still correct). It requires 378 primes of size 29 bits, takes about 1 day, requires 3 GB of memory on 1 processor, while with 6 processors it takes 6 h (requires 6 GB). The answer has integer coefficients of about 1600 digits (and not 800 as stated in J.-C. Faugère F4 article), for a little more than 1 million monomials, that's about 1.4 GB of RAM.

- The deterministic modular algorithm is much faster than Singular for Cyclic examples, and as fast for Katsura examples.
- For the random last example, the speed is comparable between Magma and Giac. This is where there are less pairs reducing to 0 (“F4remake” is not as efficient as for Cyclic or Katsura) and larger coefficients. This could suggest that advanced algorithms like F4/F5/etc. are probably not much more efficient than Buchberger algorithm for these kind of inputs without symmetries.
- Certification is the most time-consuming part of the process (except for Cyclic8). Integer certification is significantly faster than modular certification for Cyclic examples, and almost as fast for Katsura.
- We would like to stress that a computer assisted **mathematical proof** can not be performed with a closed-source software without a certification step. Therefore the relevant timings for comparison with closed-source pieces of software is the probabilistic check.

Acknowledgments. The first author thanks Michael Borchers and Zbyněk Konečný for the test cases in benchmarking the embeddable computer algebra systems. The second author wishes to thank Vanessa Vitse for insightful discussions and Frédéric Han for testing.

References

1. Ancsin, G., Hohenwarter, M., Kovács, Z.: GeoGebra goes mobile. *Electron. J. Math. Technol.* **5**(2), 160–168 (2011)
2. Arnold, E.A.: Modular algorithms for computing Gröbner bases. *J. Symbolic Comput.* **35**(4), 403–419 (2003)
3. Buchberger, B.: Gröbner bases: an algorithmic method in polynomial ideal theory. In: Bose, N.K. (ed.) *Multidimensional Systems Theory*, pp. 184–232. Reidel Publishing Company, Dordrecht (1985)
4. Dersch, H.: Jasymca 2.0 – symbolic calculator for Java, Mar 2009. <http://webuser.hs-furtwangen.de/~dersch/jasymca2/Jasymca2en/Jasymca2en.html>
5. Faugère, J.-C.: A new efficient algorithm for computing Gröbner bases (F4). *J. Pure Appl. Algebra* **139**(1–3), 61–88 (1999)
6. Faugère, J.-C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In: *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, ISSAC 2002*, pp. 75–83. ACM, New York (2002)
7. Gebauer, R., Möller, H.M.: On an installation of Buchberger’s algorithm. *J. Symbolic Comput.* **6**(2–3), 275–286 (1988)
8. Hanselman, S.: JavaScript is assembly language for the web: semantic markup is dead! Clean vs. machine-coded HTML (2011). <http://goo.gl/YKiO6B>. Accessed 8 Jan 2014

9. Hearn, A.C.: REDUCE User's Manual Version 3.8, Feb 2004. <http://reduce-algebra.com/docs/reduce.pdf>
10. Hicklin, J., Moler, C., Webb, P., Boisvert, R.F., Miller, B., Pozo, R., Remington, K.: JAMA: JAVa MAtrix package, Nov 2012. <http://math.nist.gov/javanumerics/jama/>
11. Joux, A., Vitse, V.: A variant of the F4 algorithm. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 356–375. Springer, Heidelberg (2011)
12. Kosan, T.: JSLisp. <http://sourceforge.net/p/reduce-algebra/code/HEAD/tree/trunk/jslisp>
13. Kosan, T.: MathPiper, Jan 2011. <http://www.mathpiper.org>
14. Kovács, Z.: Definition of a parabola as a locus. GeoGebraTube material (2012). <http://www.geogebraTube.org/student/m23662>
15. Kovács, Z.: GeoGebra developers' Trac wiki: theorem proving planning (2012). <http://dev.geogebra.org/trac/wiki/TheoremProvingPlanning>. Accessed 8 Jan 2014
16. Kovács, Z.: GeoGebraWeb offers CAS functionality, May 2012. <http://blog.geogebra.org/2012/05/geogebraweb-cas/>
17. Monagan, M., Pearce, R.: Sparse polynomial division using a heap. *J. Symbolic Comput.* **46**(7), 807–822 (2011)
18. Parisse, B., Graeve, R.D.: Giac/Xcas computer algebra system (2013). <http://www-fourier.ujf-grenoble.fr/~parisse/giac.fr.html>
19. Steel, A.: Gröbner basis timings page (2004). <http://magma.maths.usyd.edu.au/~allan/gb/>
20. Wikipedia. Equation solving — Wikipedia, the free encyclopedia (2013). http://en.wikipedia.org/w/index.php?title=Equation_solving&oldid=580349875. Accessed 14 Jan 2014
21. Wikipedia. Google Native Client — Wikipedia, the free encyclopedia (2013). http://en.wikipedia.org/w/index.php?title=Google_Native_Client&oldid=588335015. Accessed 8 Jan 2014
22. Wikipedia. JavaScript engine — Wikipedia, the free encyclopedia (2013). http://en.wikipedia.org/w/index.php?title=JavaScript_engine&oldid=586802475. Accessed 8 Jan 2014
23. Wikipedia. Dart (programming language) — Wikipedia, the free encyclopedia (2014). [http://en.wikipedia.org/w/index.php?title=Dart_\(programming_language\)&oldid=589448479](http://en.wikipedia.org/w/index.php?title=Dart_(programming_language)&oldid=589448479). Accessed 8 Jan 2014