Software Exhibition Abstracts ISSAC 2007

July 29 – August 1, 2007
The University of Waterloo
Waterloo, Ontario
Canada
`http://www.cs.uwaterloo.ca/issac2007`

Following are the abstracts of the Software Exhibition Session at the 2007 ACM International Symposium on Symbolic and Algebraic Computation. The Software Exhibition Chair was Hirokazu Anai (Fujitsu Laboratories Ltd. Japan).

ISSAC 2007 organized sessions to exhibit software packages produced by academic developers. These sessions are intended to promote software development activities in all areas of symbolic mathematical computation.

DISCOVERER: A tool for solving semi-algebraic systems*

Bican Xia

LMAM & School of Mathematical Sciences
Peking University
Beijing, 100871, China
E-mail: `xbc@math.pku.edu.cn`

A system in the following form

$$\begin{cases} p_1(\mathbf{u},\mathbf{x})=0,\cdots,p_s(\mathbf{u},\mathbf{x})=0,\\ g_1(\mathbf{u},\mathbf{x})\geq 0,\cdots,g_r(\mathbf{u},\mathbf{x})\geq 0,\\ g_{r+1}(\mathbf{u},\mathbf{x})>0,\cdots,g_t(\mathbf{u},\mathbf{x})>0,\\ h_1(\mathbf{u},\mathbf{x})\neq 0,\cdots,h_m(\mathbf{u},\mathbf{x})\neq 0,\end{cases}$$

is called a *semi-algebraic system*, or simply SAS, which can be also written as

$$[P,G_1,G_2,H] \tag{1}$$

where P,G_1,G_2 and H stand for $[p_1,\cdots,p_s]$, $[g_1,\cdots,g_r]$, $[g_{r+1},\cdots,g_t]$ and $[h_1,\cdots,h_m]$, respectively. Herein, $\mathbf{x}=(x_1,\cdots,x_n)$ are variables and $\mathbf{u}=(u_1,\cdots,u_d)$ are parameters whose values are in $\mathbb{R}$ and p_i, g_j, h_k are polynomials in $\mathbb{Q}[\mathbf{u},\mathbf{x}]$ with $n,s\geq 1$, $d,r,t,m\geq 0$.

An SAS is called a *constant semi-algebraic system* if it contains no parameters, *i.e.*, $d=0$, otherwise a *parametric semi-algebraic system*. We are interested in the following problems concerning the *real solutions* of an SAS S. If S is a constant SAS,

1.1 Does S have infinite number of real solutions?

1.2 If S has a finite number of (distinct) real solutions, what is the number and how to isolate the real solutions?

If S is a parametric SAS,

2.1 What condition must the parameters satisfy such that the dimension of real solutions of S is positive? And what is the dimension?

2.2 If the dimension is zero, what is the necessary and sufficient condition on the parameters for S to have a prescribed number of (distinct) real solutions?

DISCOVERER is a package† developed by the author using Maple, which can solve the problems listed above [3, 4, 5, 6, 7, 8, 9] automatically. The prerequisite to run the package is Maple 7.0 or a later version of it.

*This work was supported by Chinese NKBRPC-2004CB318003, Chinese NKBRPC-2005CB321902 and Chinese NSFC-60573007.

†The package and documentations can be downloaded at `http://www.is.pku.edu.cn/~xbc/DISCOVERER.html`. Or, one can send an email to the author asking for a copy.

The main features of DISCOVERER include *real solution classification of parametric* SASs (Problems 2.1 and 2.2), and *real solution counting and isolating of constant* SASs (Problems 1.1 and 1.2). For a parametric SAS S of the form (1) and a given non-negative integer N, to determine the necessary and sufficient conditions on $\mathbf{u}$ such that the number of distinct real solutions of S equals N, one can first call

$$\texttt{tofind}(P, G_1, G_2, H, [\mathbf{x}], [\mathbf{u}], N).$$

The output of `tofind` is a quantifier-free formula Φ in parameters[‡] and a *border polynomial* $\mathrm{BP}(\mathbf{u})$ which mean that, provided $\mathrm{BP}(\mathbf{u}) \neq 0$, the necessary and sufficient condition for S to have exactly N distinct real solutions is Φ holds.

Then, letting $P' = [\mathrm{BP}, p_1, ..., p_s]$, one can call

$$\texttt{Tofind}(P', G_1, G_2, H, [\mathbf{x}], [\mathbf{u}], N)$$

to find conditions on $\mathbf{u}$ when the parameters are on the "boundary" $\mathrm{BP}(\mathbf{u}) = 0$.

If the argument N is a range $b..c$ where b and c are non-negative integers and $b < c$ or a range $b..w$ where b is a non-negative integer and w is a name without value, standing for a large enough integer, the program obtains necessary and sufficient conditions such that the number of distinct real solutions of S falls in the range. Especially, calling

$$\texttt{tofind}(P, G_1, G_2, H, [\mathbf{x}], [\mathbf{u}], 1..w),$$

is equivalent to performing quantifier elimination to $\exists x_1 \cdots \exists x_n\ (S)$ where S is viewed as conjunction of atomic formulae. At this point, the functions of DISCOVERER overlap some famous tools such as QEPCAD[1] and REDLOG [2].

For a constant SAS S (*i.e.*, $d = 0$) of the form (1), if S has only a finite number of real solutions, DISCOVERER can determine the number of distinct real solutions of S, say M, and moreover, can find out M disjoint cubes with rational vertices in each of which there is only one solution. In addition, the width of the cubes can be less than any given positive real. The two functions are realized through calling

$$\texttt{nearsolve}(P, G_1, G_2, H, [\mathbf{x}]) \ \text{ and } \ \texttt{realzeros}(P, G_1, G_2, H, [\mathbf{x}], w),$$

respectively, where w is optional, which indicates the maximum size of the output cubes.

There are some other functions of DISCOVERER that we can not introduce here due to page limitation, which include *relatively simplicial decomposition* of a triangular set with respect to a polynomial, *discriminant sequences* of polynomials, *negative-root discriminant sequences* of polynomials, and so on. For a detailed description of these concepts, please refer to the references listed below. For a demo on how to use the functions of DISCOVERER, please refer to the "Demo-DISCOVERER.mws" file which can be downloaded together with the package.

References

[1] G. E. Collins and H. Hong: Partial cylindrical algebraic decomposition for quantifier elimination, *J. Symb. Comput.*, **12**: 299–328, 1991.

[2] A. Dolzman and T. Sturm: REDLOG: Computer algebra meets computer logic. *ACM SIGSAM Bulletin*, **31**(2): 2–9, 1997.

[3] B. Xia and X. Hou: A Complete Algorithm for Counting Real Solutions of Polynomial Systems of Equations and Inequalities. *Comput. Math. Appl.*, **44**: 633–642, 2002.

[4] B. Xia and L. Yang: An algorithm for isolating the real solutions of semi-algebraic systems. *J. Symb. Comput.,* **34**: 461–477, 2002.

[5] B. Xia and T. Zhang: Real Solution Isolation Using Interval Arithmetic. *Comput. Math. Appl.*, **52**: 853–860, 2006.

[6] L. Yang, X. Hou and B. Xia: Automated Discovering and Proving for Geometric Inequalities. In: *Proc. ADG'98 —- LNAI 1669*, Springer-Verlag, 30–46, 1999.

[7] L. Yang, X. Hou and B. Xia: A complete algorithm for automated discovering of a class of inequality-type theorems. *Sci. China (Ser. F)* **44**: 33–49 (2001).

[8] L. Yang and B. Xia: Automated Deduction in Real Geometry. In: *Geometric Computation*, (F. Chen and D. Wang, eds.), World Scientific, 248–298, 2004.

[9] L. Yang and B. Xia: Real solution classifications of a class of parametric semi-algebraic systems. In *Algorithmic Algebra and Logic —- Proc. A3L 2005* (A. Dolzmann, A. Seidl, and T. Sturm, eds.), 281–289, 2005.

[‡]Probably with some variables if the dimension of complex solutions of the equations is positive.

PGB: A package for computing parametric Gröbner and related objects

Katsusuke Nabeshima

Research Institute for Symbolic Computation (RISC-Linz),
Johannes Kepler University, Linz, A-4040, Austria
Katsusuke.Nabeshima@risc.uni-linz.ac.at

1 Introduction

We present a new software package in the computer algebra system Risa/Asir [NT92], named PGB, for computing parametric Gröbner bases in various domains and related objects. A parametric Gröbner basis is a Gröbner basis for a polynomial ideal with parametric coefficients. One can download the package PGB from the following website.
http://www.risc.uni-linz.ac.at/research/compalg/software/

2 PGB: A Package for Computing Parametric GB

The package PGB is for computing CGBs (comprehensive Gröbner bases) and CGSs (comprehensive Gröbner systems) [Wei92, SS06, Nab07] in various domains. Let $\mathbb{Q}$ be the field of rational numbers, $\bar{A} := \{A_1,\dots,A_m\}$ parameters and $\bar{X} := \{X_1,\dots,X_n\}$ variables. There exist some implementations for computing parametric Gröbner bases, namely the Maple implementation* by Manubens and Montes [Mon02, MM06], the Reduce implementation† by Dolzmann and Sturm [DS97], the Risa/Asir implementation‡ by Suzuki and Mathematica implementation by Nagasaka. These implementations work only on a commutative polynomial ring $(\mathbb{Q}[\bar{A}])[\bar{X}]$. The package PGB works on $(\mathbb{Q}[\bar{A}])[\bar{X}]$, rings of differential operators and $(\mathbb{Q}[\bar{A}])[\bar{X}]$-modules. Moreover, the new algorithm [Nab07] for computing comprehensive Gröbner systems, have been implemented in the package.

2.1 CGBs and CGSs in $(\mathbb{Q}[\bar{A}])[\bar{X}]$ (normal case)

There exist many commands in the package, for computing CGSs and CGBs in $(\mathbb{Q}[\bar{A}])[\bar{X}]$ (normal case). Some of the commands follow the Suzuki-Sato algorithm [SS06], and some of the commands follow the new algorithm [Nab07].

2.2 CGBs and CGSs in rings of differential operators

By using the package PGB, we can compute CGSs and CGBs in rings of differential operators. That is, let $\mathbb{Q}[\bar{X}]$ be a ring of polynomials in n variables over $\mathbb{Q}$, $\partial_i = \frac{\partial}{\partial x_i} : \mathbb{Q}[\bar{X}] \to \mathbb{Q}[\bar{X}]$ the partial derivative by x_i, $1 \le i \le n$ and $\mathbb{Q}[\bar{X},\bar{D}] := \mathbb{Q}[x_1,\dots,x_n,\partial_1,\dots,\partial_n]$. $\mathbb{Q}[\bar{X},\bar{D}]$ has the commutation rules $x_ix_j = x_jx_i$, $\partial_i\partial_j = \partial_j\partial_i$, $\partial_ix_j = x_j\partial_i$, for $i \neq j$, and $\partial_ix_i = x_i\partial_i + 1$. It is well-known that $\mathbb{Q}[\bar{X},\bar{D}]$ is a left-Noetherian. By the package, one can compute CGSs and CGBs in this domain.

For example, let $F = \{2\partial_1 + bx_2\partial_2 + x_2, x_1\partial_1 - ax_2\} \subset (\mathbb{Q}[a,b])[x_1,x_2,\partial_1,\partial_2]$, a,b parameters, x_1,x_2 variables and ∂_1,∂_2 partial derivative by x_1,x_2, respectively. Let $\succ$ be the graded lexicographic order such that $x_1 \succ x_2 \succ \partial_1 \succ \partial_2$. By using the package, we can compute a comprehensive Gröbner system and basis for $\langle F \rangle$ with respect to $\succ$.

2.3 CGBs and CGSs for modules

By using the package PGB, we can compute CGSs and CGBs in $(\mathbb{Q}[\bar{A}])[\bar{X}]^r$ where r is a natural number. Let x,y be variables and a,b parameters.
We have $f_1 = \begin{pmatrix} ax - bx + 1 \\ ax^2y + ax + b \end{pmatrix}$ and $f_2 = \begin{pmatrix} by + a \\ bx^2 + bx + 2 \end{pmatrix}$ in $(\mathbb{Q}[a,b])[x,y]^2$. By using the package, we can compute a comprehensive Gröbner system and basis for a submodule generated by $\{f_1, f_2\}$ with respect to a module order.

2.4 Some related objects

Computing Gröbner bases for modules are closely related to computing syzygies. Actually, in the parametric polynomial case, this relation is also the same. The command for computing parametric syzygies, has been implemented in the package. There are some useful commands in the package.

*http://www-ma2.upc.edu/~montes/
†http://students.fim.uni-passau.de/~reduce/cgb/
‡http://kurt.scitec.kobe-u.ac.jp/~sakira/CGBusingGB/

Acknowledgments.
This work has been supported by the Austrian Science Foundation (FWF), project P16357-N04. The author thanks Prof. Franz Winkler for stimulating discussion and suggestions.

References

[DS97] Dolzmann, A. and Sturm, T. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31(2):2–9, 1997.

[MM06] Manubens, M. and Montes, A. Improving DISPGB algorithm using the discriminant ideal. *Journal of Symbolic Computation*, 41:1245–1263, 2006.

[Mon02] Montes, A. A new algorithm for discussing Gröbner basis with parameters. *Journal of Symbolic Computation*, 33/1-2:183–208, 2002.

[Nab07] Nabeshima, K. A Speed-Up of the Algorithm for Computing Comprehensive Gröbner Systems. In Brown, C., editor, *International Symposium on Symbolic and Algebraic Computation*. ACM-Press, 2007. To appear.

[NT92] Noro, M. and Takeshima, T. Risa/Asir- A Computer Algebra System. In Wang, P., editor, *International Symposium on Symbolic and Algebraic Computation*, pages 387–396. ACM-Press, 1992. `http://www.math.kobe-u.ac.jp/Asir/asir.html`

[SS06] Suzuki, A. and Sato, Y. A Simple Algorithm to compute Comprehensive Gröbner Bases using Gröbner bases. In *International Symposium on Symbolic and Algebraic Computation*, pages 326–331. ACM Press, 2006.

[Wei92] Weispfenning, V. Comprehensive Gröbner bases. *Journal of Symbolic Computation*, 14/1:1–29, 1992.

SNAP package for Mathematica

Kosaku Nagasaka

Kobe University, Japan

Abstract

SNAP (Symbolic Numeric Algebra for Polynomials) package for *Mathematica* provides various functions to compute approximate algebraic properties including approximate GCD and factorization of polynomials for example. For practical situations, the package does not have enough functionalities yet. However, the aim of this package is showing how an unified tolerance mechanism that we introduced for the package works. With that, we can continue approximate calculations under certified tolerances without special skills in symbolic-numeric algebra.

Short Introduction

Recently, there are a lot of results in the area called Symbolic Numeric Algebra. However, there is not any *Mathematica* package including such results, except for our "SNAP" package for *Mathematica*, and it is an abbreviation for Symbolic Numeric **Algebra** for Polynomials. By Algebra, we mean that continuous applicability of approximate operations: for example, computing an approximate GCD between an empirical polynomial and a polynomial which is the nearest singular polynomial computed by functions included in our package, of another empirical polynomial. This continuous applicability is more important for practical computations. The package was introduced in 2004 [2] and in 2006 [3].

– Features: The package provides limited but enough functionalities to use some symbolic-numeric operations without any special knowledges in this area. Our package is for *Mathematica* and most average users of *Mathematica* are not interested in algorithms and any inner error tracking system. Therefore, we make our package usable for them: 1) seamless error tracking system with *Mathematica*'s, 2) seamless basic operations with *Mathematica*'s built-in functions, and 3) symbolic-numeric operations being compatible with 1) and 2).

– Tolerance Mechanism: We define the following empirical polynomial structures (like a set), for $f(x,y,\ldots,z) \in \mathbb{C}[x]$ and tolerance $\varepsilon \in \mathbb{R}$, in our package.

$$P_p(f,\varepsilon) = \{\tilde{f} \mid \tilde{f} \in \mathbb{C}[x],\ \deg_x \tilde{f} \leq \deg_x f,\ \|f-\tilde{f}\|_p \leq \varepsilon\},$$

where p denotes $1, 2$ or ∞. With this definition, most of functions in our package give us results satisfying the following (f and g are given and h is an output):

$$\forall \tilde{f} \in P_p(f,\varepsilon_f), \forall \tilde{g} \in P_p(g,\varepsilon_g), \exists \tilde{h} \in P_p(h,\varepsilon_h), \tilde{h} = \tilde{f} \circ \tilde{g},$$

where $\circ$ denotes each operation provided by our functions.

Implemented operations are basic three operations (addition, subtraction and multiplication) with polynomials, computing quotient and remainder of polynomials, derivatives of polynomials, substitution of variables with numerical values, changing error schemes (1,2 or ∞-norm), irreducibility testing for multivariate polynomials, separation bounds for multivariate polynomials, numerical factoring of multivariate monic polynomials, computing comprehensive Gröbner systems, co-primeness testing for univariate and multivariate polynomials, computing approximate univariate polynomial GCD, computing nearest singular polynomials and computing numerical roots with correct error bounds.

– How to Use the Package: The package is available with some documents in the *Mathematica* help document format. Here, we show some examples. The package can compute a separation bound for multivariate polynomials.

```
In[1]:= Needs["SNAP`"]
In[2]:= (x^2+y*x+2y-1)(x^3+y^2 x-y+7)+0.2x;
In[3]:= f=SetAccuracy[%,8]
```
Out[3]= $-7.0000000 + 0.2000000x + 7.0000000x^2 - 1.0000000x^3 + x^5$
$+15.0000000y + 7.0000000xy - 1.0000000x^2y + 2.0000000x^3y$
$+x^4y - 2.0000000y^2 - 2.0000000xy^2 + x^3y^2 + 2.0000000xy^3 + x^2y^3$
```
In[4]:= SeparationBound[f]
Out[4]= 0.000767686952
In[5]:= SeparationBound[f,Method->NewtonPolytope]
Out[5]= 0.00267931444
```
The package tries to solve the algebraic equation with priori error, by computing its comprehensive Gröbner system, hence, the following output is correct for the all possible coefficients of input polynomials within the priori error bound.
```
In[6]:= f1 = SNAP[2.0x-3y+1.0]; f2 = SNAP[1.0x-1.0y+1.0];
In[7]:= NSolve[{f1==0,f2==0},{x,y}]
```
Out[7]= {{{x→ $-2.000000000000 + 0. \times 10^{-13} i$,
y→ $-1.0000000000000 + 0. \times 10^{-14} i$}}}
```
In[8]:= Map[Tolerance,{x,y}/.%]
```
Out[8]= {{{1.28622×10^{-12}, 2.00972×10^{-14}}}}

– How to Get the Package: The package is available at the following URL:
`http://wwwmain.h.kobe-u.ac.jp/~nagasaka/research/snap/`

References

[1] R. J. Fateman. A review of mathematica. *J. Symbolic Comput.*, 13:545–579, 1992.

[2] K. Nagasaka. Snap package for mathematica and its applications. In *Proc. The Ninth Asian Technology Conference in Mathematics*, pages 308–316, 2004.

[3] K. Nagasaka. Using coefficient-wise tolerance in symbolic-numeric algorithms for polynomials. *J. JSSAC*, 12(3):21–30, 2006.

[4] M. Sofroniou and G. Spaletta. Precise numerical computation. *Journal of Logic and Algebraic Programming*, 64(1):113–134, 2005.

[5] M. Trott. *The Mathematica Guide Book for Numerics*. Springer, 2005.

Implementation of CGS and CGB on Risa/Asir and other computer algebra systems using Suzuki-Sato algorithm

Akira Suzuki[1] Yosuke Sato[2]

1. Kobe University, Japan, sakira@kurt.scitec.kobe-u.ac.jp
2. Tokyo University of Science, Japan, ysato@rs.kagu.tus.ac.jp

In [5], we showed simple algorithms to compute comprehensive Gröbner systems (CGS) and comprehensive Gröbner bases (CGB). We call them Suzuki-Sato algorithms in this paper. Other known existing software packages to compute CGS or CGB are CGB of Redlog (Weispfenning) [1], DISPGB, and MCCGS (Montes) [2, 3].

First advantage of our software is on its simplicity. The above other software packages use very complicated algorithms. They are essentially based on a Buchberger algorithm in a polynomial ring over a coefficient field of rational functions of parameters together with rather complicated conditions of parameters. Meanwhile, our algorithms require only computations of reduced Gröbner bases in a polynomial ring over a ground field. This simplicity makes it extremely easy to implement our algorithm on many computer algebra systems that have a routine to compute usual reduced Gröbner bases. We actually implemented them on Risa/Asir[§] [4], Singular, Maple, and Mathematica. All of them are available from our Web site.[¶]

The simplicity of our algorithms also gives us the second advantage. Since the main portion of our software is the computation of usual reduced Gröbner bases, we can directly use the performance of the built-in Gröbner bases computation package. As a consequence, our implementation on Risa/Asir which has a fast Gröbner bases computation package is much faster than the others in case we do not have many parameters. It can be run even on a small PDA machine such as Zaurus.

On the other hand, we should mention that our software is not so fast when we have more parameters than main variables. In some case, it is slower than the others. Another disadvantage of our software is on the size of the outputs. In some cases, the outputs is much bigger than the others.

We focus on our implementation on Risa/Asir. Risa/Asir has not only a fast and sophisticated Gröbner bases computation package but also has many useful options. These options enable us to develop more flexible implementation than the implementations on the other computer algebra systems.

We can use the command `cgs()` for a CGS which is a straightforward implementation of the original algorithm appearing in the paper, `rcgs()` for a reduced CGS, and `cgb()` for a faithful CGS. We show an example to use these command. After you start Risa/Asir, you should input two commands "`load("gr");`" and "`load("primdec");`" since our implementations require the two packages for Gröbner bases and primary ideal decompositions. (Though the package for Gröbner bases are always required, the one for primary ideal decomposition will be used only when we use the option `primdec`.) After then, if we input the command "`load("`*an_appropriate_path*`/acgs.rr");`", our implementation would be ready to compute CGS and CGB.

For example, let us calculate a reduced CGS of $F = \{X^3 - A, Y^4 - B, X + Y - Z\}$ with respect to the lexicographic term order with $X \gg Y \gg Z$ where A and B are parameters. For it, we first set the variable F by typing "`F = [x^3-a,y^4-b,x+y-z];`". To calculate the reduced CGS for F, we type "`G = acgs.rcgs(F,[x,y,z],2);`" where `[x,y,z]` assigns the variable (and the remained indeterminants `a` and `b` are interpreted as parameters) and `2` set the lexicographic term order. The input parameters of this routine is a subset of one of the built-in reduced Gröbner basis command `gr()`, thus you can refer the Asir User's Manual[‖] for more details on these parameters. We should mention that the output of this command `rcgs()` is in a kind of internal expression of CGS, and so we should use an additional command for our recognizing it. By typing "`acgs.bases2str(G);`", we can get a human-readable expression of `G`, a reduced CGS for F.

Next, let us calculate a faithful CGS and a CGB of the same F as in the previous paragraph. By typing "`FG = acgs.cgb(F,[x,y,z],2);`", we can assign a faithful CGS for F to the variable `FG` in a kind of internal expression as above. As you can guess, the input parameters of the command `cgb()` is the same as one of `rcgs()`. From the internal expression `FG`, we get the human-readable output for faithful CGS by typing "`acgs.bases2str(FG);`" and one for CGB by typing "`acgs.bases_subst(FG);`".

Our software also has many usefull optional switches, such as `zradical`, `redinput`, `primdec`, `hgr`, `elimdup`, and `interred` in version 1.3a. In the exhibition, we give a brief description and a demonstration using them.

References

[1] Dolzmann, A. and Sturm, T. (1997). Redlog: Computer algebra meets computer logic, ACM SIGSAM Bulletin, 31, 2, 2–9.

[§] http://www.math.kobe-u.ac.jp/Asir/asir.html
[¶] http://kurt.scitec.kobe-u.ac.jp/~sakira/CGBusingGB/
[‖] http://www.math.kobe-u.ac.jp/OpenXM/Current/doc/asir2000/html-eg/man_toc.html

[2] Manubens, M. and Montes, A. (2004). Improving DISPGB Algorithm using the discriminant ideal, J. Symb. Comp., 41, 1245–1263.

[3] Manubens, M. and Montes, A. (2006). Minimal Canonical Comprehensive Gröbner System, preprint MA2-IR-06-00015.

[4] Noro, M. and Takeshima, T. (1992). Risa/Asir – A Computer Algebra System. International Symposium on Symbolic and Algebraic Computation (ISSAC 92), Proceedings, 387–396.

[5] Suzuki, A. and Sato. Y. (2006). A Simple Algorithm to Compute Comprehensive Gröbner Bases Using Gröbner Bases, International Symposium on Symbolic and Algebraic Computation (ISSAC 2006), Proceedings, 326–331.

GAP – Groups, Algorithms, Programming

Steve Linton

The GAP Group

GAP is a system for computational discrete algebra, with particular emphasis on Computational Group Theory. GAP provides a programming language, a library of thousands of functions implementing algebraic algorithms written in the GAP language, large data libraries of algebraic objects and a growing collection of user-contributed extension packages. GAP is widely used in research and teaching for studying groups and their representations, rings, vector spaces, algebras, combinatorial structures, codes, Lie algebras and more. It has been cited in over 800 published papers. The system, including source, is distributed freely. You can study and easily modify or extend it for your special use.

The GAP website is found at `http://www.gap-system.org`. From there you can download GAP, read and search the documentation and browse a collection of talks, course notes, worked examples and other information.

I will be happy to discuss and demonstrate any aspect of the system at ISSAC.

1 Capabilities

GAP provides:

- Mathematical capabilities accessible through
  - a large library of functions, containing implementations of various algebraic algorithms, part of which is divided into 'Modules' under the responsibility of 'maintainers',
  - 70 separate packages of additional functions for specialized purposes which can be used like library functions,
  - data libraries containing large classes of various algebraic objects that are accessible by using GAP commands.
- A programming language, also called GAP, which is interpreted and can be compiled. It can be used interactively at the keyboard or to write programs to be saved and then executed. Such programs can easily be modified and rerun. The language features:
  - Pascal-like control structures,
  - automatic memory management including garbage collection,
  - streams,
  - flexible list and record data types,
  - built-in data types for key algebraic objects,
  - automatic method selection building on a mechanism for automatically choosing the highest ranked method for a certain operation, depending on the current state of all its arguments, so that GAP objects representing mathematical objects may gain knowledge about themselves during their lifetime resulting in better methods being chosen later on.
- An interactive environment that supports in particular
  - line editing e.g. tab completion,
  - break loops for debugging,

  - further debugging and profiling facilities for GAP programs,
  - online help (i.e. online access to the manuals),
  - a graphical user interface for GAP(UNIX only),
  - further GAP interface programs provided by users.
- Documentation, in particular
  - a Tutorial and further learning and teaching material, some of these in French, Japanese, Portuguese, and Russian,
  - the Reference Manual giving complete descriptions of library functions with examples of use,
  - separate manuals for the packages,
  - worked out higher level examples and a collection of preprints and talks,
  - advice for people writing GAP code,
  - a mark-up language GAPDoc for writing GAP documentation,
  - an archive of GAP Forum contributions,
  - a bibliography of papers quoting GAP.
- Download and Installation instructions.

2 Main Mathematical Areas

- **G**roup Theory: generic group functionality, permutation and matrix groups, finitely presented groups, representationas and characters of groups
- **A**lgebra: Vector spaces, modules, algebras especially Lie algebras, semigroups, monoids and near-rings
- **Combinatorics:** graphs, codes and designs
- Words, rewriting and automata

3 Requirements and Availability

The GAP system will run on any machine with a UNIX-like or recent Windows or MacOS operating system although some packages are only available for UNIX-like systems. GAP is free software under the GP (version 2).

ALLTYPES in the Web

Fritz Schwarz

FhG, Institute SCAI, 53754 Sankt Augustin, Germany
Email: fritz.schwarz@scai.fhg.de,
URL: www.scai.fhg.de/schwarz.html

ALLTYPES which abbreviates *ALgebraic Language and TYPe System* provides an interactive environment in the internet that is particularly well suited for performing symbolic computations in differential algebra. Its novel design has been guided by the following principles.

- The system should be readily available without any installation or maintenance work.
- It should be easy to use without the need of reading voluminous manuals.

In more detail, the most important features for achieving this goal may be described as follows:

- A set of about fifty parametrized algebraic types is defined.
- Data objects represented by these types may be manipulated by more than one hundred polymorphic functions.
- Reusability of code is achieved by genericity and inheritance. The user may extend the system by defining new types and polymorphic functions.

- A language comprising seven basic language constructs is defined for implementing mathematical algorithms efficiently.
- The manipulation of types is supported in ALLTYPES by a special portion of the language.
- Currently the userinterface provides about fifty functions for working in commutative and differential algebra.

The website `www.alltypes.de` makes the interactive userinterface available in the internet. Its main advantage is that it may be used without any prior knowledge of the system, and no installation work and periodic updating is necessary. In detail its most important features are:

- The complete documentation that is needed for working with the userinterface functions is part of the website.
- Several demos and tutorials make it easy to become familiar with the system.
- An interactive session may be started in a pop-up window in order to apply the userinterface functions.
- There are databases with ordinary and partial differential equations and relevant literature.

The functionality of the ALLTYPES website has been presented in a short demo at the ISSAC'07 in Waterloo.

A Maple Library for High Performance Sparse Polynomial Arithmetic

Roman Pearce Michael Monagan

Simon Fraser University

We have developed a library for sparse polynomial arithmetic to supplement Maple's internal routines. Along the way we identified three common sources of inefficiency in existing implementations.

The first is memory allocation. Allocating memory for every term that may appear in a computation produces significant overhead. If, as in exact division, terms introduced by one step of a computation are cancelled off in a later step this time is effectively wasted.

The second is cache performance. The L1 and L2 caches of modern processors run at the same speed as the processor whereas main memory is 4-8 times slower. The time to access main memory may be 100 CPU cycles versus 1-20 cycles for cache. Data structures which access memory randomly, such as linked lists, exhibit poor performance when the size of the polynomials exceeds the size of the cache.

A third source of inefficiency is the number of monomial comparisons. Consider the division of a sparse polynomial with nm terms by a divisor with m terms producing a quotient with n terms. If we repeatedly subtract multiples of the divisor using merges we may do $O(n^2m)$ monomial comparisons. Singular and CoCoA implement "geobuckets" to reduce the number of comparisons to $O(nm \log nm)$. One problem with geobuckets is that $O(nm)$ memory is used for division even though the size of the result is only $O(n)$.

In our library we store polynomials as large arrays of dense exponent vectors followed by coefficients. To improve performance and decrease memory usage we have the option to pack multiple exponents into each machine word. The use of arrays eliminates most cache misses when terms are accessed sequentially. To amortize the cost of memory allocation we acquire large blocks of memory at a time, and reuse this storage in subsequent calls to the algorithms.

To implement division, multiplication and n-ary addition we do not merge polynomials one-by-one into an intermediate object. Instead, we construct a heap of pointers into the input and use the heap to do a simultaneous n-ary merge. For example, to multiply two polynomials f and g we would maintain a heap of $\#f$ elements each pointing to a different term of f and containing an associated pointer which increments along g. The algorithm repeatedly extracts the largest unmerged $f_i g_j$ and inserts the next term $f_i g_{j+1}$ from the partial product $f_i g$.

A heap of pointers has good complexity. Multiplication of polynomials with n and m terms does $O(nm \log(min(n,m)))$ monomial comparisons – the same as divide-and-conquer. For division where the quotient has n terms and the divisor has m terms the number of comparisons is $O(nm \log n)$. This is asymptotically better than geobuckets, however geobuckets often do fewer monomial comparisons in practice.

The main advantage of using a heap is the memory usage, which is the size of the heap plus the size of the result. For exact division we use $O(n)$ memory, where n is the size of the quotient. Because a heap of pointers is small and we do not construct any large intermediate objects, the algorithms run almost entirely in the processor cache. This makes the heap algorithms perform better than geobuckets for large problems in practice.

One more advantage of using a heap is that coefficient arithmetic is delayed until we need to do it. Terms are merged by descending monomial, so when a test for division fails we have merged the minimal number of terms required to discover that fact.

We have also constructed a variant of the heap division algorithm where the size of the heap is equal to the size of the divisor(s). Its complexity is $O(nm \log m)$, where n is the size of the quotient(s) and m is the size of the divisor(s), so it is asymptotically faster than previous division algorithms when the divisor(s) are smaller than the quotient(s). Using this algorithm, we can reduce large polynomials of very high degree by a small set of algebraic extensions in time proportional to the number of reducible terms encountered.

Our C library is still under development and is expected to be part of a future release of Maple. In the meantime, development versions can be downloaded from `http://www.cecm.sfu.ca/~rpearcea`.

CoCoA: Computations in Commutative Algebra

John Abbott[1] Anna Bigatti[1] Massimo Caboara[2] Lorenzo Robbiano[1]

1. Università di Genova 2. Università di Pisa

What is CoCoA?

CoCoA is a *special-purpose* system for doing **Co**mputations in **Co**mmutative **A**lgebra. It runs on all common platforms.

CoCoA is *freely available* software for research and educational purposes: the latest version is *CoCoA 4.7.3* (September 2007) which can be obtained from

`http://cocoa.dima.unige.it/`

The system offers a textual interface, an Emacs mode, and a graphical user interface common to most platforms.

The main features of CoCoA

Aside from computing Gröbner bases, CoCoA's particular strengths include ideal and module operations (such as syzygies and minimal free resolutions, intersections, divisions, and the radical of an ideal), polynomial factorization, exact linear algebra, Hilbert functions, zero-dimensional schemes and toric ideals.

The usefulness of these technical skills is enhanced by the mathematically natural language for describing computations. This language is readily learned by students, and enables researchers to explore and develop new algorithms without the administrative tedium necessary when using "low-level" languages.

The users of CoCoA

Currently CoCoA is used by researchers in several countries. Most of them are Commutative Algebraists and Algebraic Geometers, but also people working in other areas such as Analysis and Statistics have already benefitted from our system.

Thanks to recent collaboration on approximate computation in commutative algebra there is now the extension ApCoCoA (see `http://www.apcocoa.org/`) useful for certain "real-world" computations.

CoCoA places great emphasis on being easy and natural to use, so it is also used as the main system for teaching advanced courses in several universities in Europe and North America. It is mentioned in some of the most widely used text books in computational algebra, and plays a major role in the book by Kreuzer & Robbiano "Computational Commutative Algebra" (Springer).

The future of CoCoA

Lately the CoCoA project has entered a new phase: completely rebuilding and restructuring the program employing more modern algorithms where available. The aims of the new version include offering better flexibility and performance while retaining the simplicity of use for which CoCoA has become widely appreciated.

A crucial design decision was the passage from C to C++ as the implementation language: the improved expressivity of C++ allows the source code to be more readable while offering better run-time performance. We expect concomitant benefits for future maintenance of the source. The new design is expressly as a **library**; a **server** (communicating via OpenMath) and a standalone interactive **system** are being built on top of this library.

CoCoALib, being the core of the project, is also its most evolved part, and is the part that we shall look at most closely. In keeping with the theme of ready accessibility the software is free and open in the sense of the GPL.

Philosophy of CoCoALib

Our aim is for CoCoALib to offer reference implementations of the principal algorithms in computational commutative algebra. The development of the library and the other components requires an enormous investment of time and resources. So that this investment is worthwhile we want to ensure that the software is widely available and will live for a long period of time. Consequently our implementations have to satisfy various design criteria:

∗ the code must exhibit good run-time performance
∗ the source code must be clear and well designed
∗ the source code must be well documented (both for users and maintainers)
∗ the source code must be clean and portable
∗ the code must be easy and natural to use

One implication of these criteria is that the design should reflect the underlying mathematical structure since this will ensure that the library is natural to use.

Another implication is that we regard clear and comprehensible code as being generally more desirable than arcanely convoluted code striving to achieve the utmost in run-time performance, because clear code is easier to maintain and should live longer. Our experience is showing that this emphasis on cleanliness is also providing quite good run-time performance.

SyNRAC: A Maple Toolbox for Solving Real Algebraic Constraints

Hitoshi Yanami[1,2] Hirokazu Anai[1,2]

1. Information Technology Core Laboratories,
Fujitsu Laboratories Ltd.
Kamikodanaka 4-1-1, Nakahara-ku, Kawasaki 211-8588, Japan
yanami@flab.fujitsu.co.jp, anai@jp.fujitsu.com

2. CREST, Japan Science and Technology Agency
Kawaguchi Center Building, 4-1-8, Honcho, Kawaguchi 332-0012, Japan

Abstract

We introduce various aspects of design and implementation of a symbolic computation toolbox, called SyNRAC, handling first-order formulas on top of Maple. SyNRAC provides us with a new set of tools for solving real algebraic constraints derived from a broad range of applications in science and engineering.

1 SyNRAC on Maple

SyNRAC is a package of commands written in the computer algebra system Maple. This package indeed provides an environment for dealing with first-order formulas over the reals [1, 2, 3]. SyNRAC stands for a Symbolic-Numeric toolbox for Real Algebraic Constraints.

We have started its development with a focus on the implementation of symbolic quantifier elimination (QE). The development began in 2002 and it first appeared in a literature in 2003 [1]. The solvers to be addressed include mainly symbolic ones and also symbolic-numeric ones to improve the efficiency of the symbolic approach. One of the big advantages of this approach is that they can deal with parametric and nonconvex constraints.

The focus of the implemented algorithms is on practically effective quantifier elimination for certain industrial/engineering problems and simplification of quantifier-free formulas. Therefore SyNRAC provides a yet another implementation of quantifier elimination, which is still missing in Maple.

To handle problems on first-order formulas, a particular environment should be prepared. Crucial points of our implementation are as follows:

- **logical operations:** We have made efforts for extending symbolic computation from computer algebra to first-order logic on Maple,
- **algebraic extensions:** We definitely need to establish how computations over algebraic number fields are realized (data types, their efficient representation, etc).

Among various computer algebra systems we have chosen Maple as a platform. The reason lies mainly in the following:

- Maple-packages are automatically incorporated into MATLAB, which is widely used in engineering, via its "Symbolic Math Toolbox,"
- they provide a good environment to realize symbolic-numeric solvers such as floating-point arithmetic and many numerical packages for optimization.

Currently the following algorithms are available in SyNRAC:

- special QE by the Sturm-Habicht sequence for sign definite condition,
- special QE by virtual substitution for linear and quadratic formulas,
- general QE by cylindrical algebraic decomposition,
- simplification of quantifier-free formulas.

2 Applications

A broad range of applications in science and engineering can be expressed in a framework of first-order logic [4]. Therefore we also aim to make SyNRAC a comprehensive toolbox composed of a collection of solvers for real algebraic constraints, described by first-order formulas that are derived from various engineering problems.

We have, for instance, successfully developed a toolbox tailored for a specific application field based on SyNRAC , e.g., robust control design, on MATLAB, which would be a novel tool that provides new systematic design procedures for engineers [5].

References

[1] Anai, H., Yanami, H.: SyNRAC: A maple-package for solving real algebraic constraints. In: Proceedings of International Workshop on Computer Algebra Systems and their Applications (CASA) 2003 (Saint Petersburg, Russian Federation), P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2657, Springer (2003) 828–837

[2] Yanami, H., Anai, H.: Development of SyNRAC—formula description and new functions. In: Proceedings of International Workshop on Computer Algebra Systems and their Applications (CASA) 2004 : ICCS 2004, LNCS 3039, Springer (2004) 286–294

[3] Yanami, H., Anai, H.: Development of SyNRAC. In: International Conference on Computational Science (3). (2005) 602–610

[4] Sturm, T.: New domains for applied quantifier elimination. In: CASC. (2006) 295–301

[5] Anai, H., Yanami, H., Sakabe, K., Hara, S.: Fixed-structure robust controller synthesis based on symbolic-numeric computation: design algorithms with a cacsd toolbox (invited paper). In: Proceedings of CCA/ISIC/CACSD 2004 (Taipei, Taiwan). (2004) 1540–1545

AXIOM – Open Source Computer Algebra System

William S. Page

Axiom Developer
bill.page@newsynthesis.org

Abstract

Axiom has been in development since 1971. Originally called Scratchpad II, it was developed by IBM under the direction of Richard Jenks[1]. The project evolved over a period of 20 years as a research platform for developing new ideas in computational mathematics. ScratchPad also attracked the interest and contributions of a large number of mathematicians and computer scientists outside of IBM. In the 1990s, the Scratchpad project was renamed to Axiom, and sold to the Numerical Algorithms Group (NAG) in England who marketed it as a commercial system. NAG withdrew Axiom from the market in October 2001 and agreed to release Axiom as free software, under an open source license.

Tim Daly (a former ScratchPad developer at IBM) setup a pubic open source Axiom project[2] in October 2002 with a primary goal to improve the documentation of Axiom through the extensive use of literate programming[3]. The first free open source version of Axiom was released in 2003. Since that time the project has attracted a small but very active group of developers and a growing number of users.

This exhibit includes a laptop computer running a recent version of Axiom, Internet access (if available) to the Axiom Wiki website[4], and CDs containing Axiom software for free distribution[5].

References

[1] Jenks, R.J. and Sutor, R.S. "Axiom – The Scientific Computation System" Springer-Verlag New York (1992) ISBN 0-387-97855-0

[2] Daly, Tim, "Axiom Computer Algebra System" http://savannah.nongnu.org/projects/axiom

[3] Knuth, Donald E., "Literate Programming" Center for the Study of Language and Information ISBN 0-937073-81-4 Stanford CA (1992)

[4] Page, William, "The Axiom Wiki Website" http://wiki.axiom-developer.org

[5] Portes, Jose Alfredo, "Doyen" http://wiki.axiom-developer.org/Doyen