Dagstuhl – Seminar
on
Computeralgebra – Software

February 05 - 09, 1996

Organizers:
Johannes Buchmann (Saarbrücken)
Rüdiger Loos (Tübingen)
Roman Mäder (Zürich)

Overview

The main focus of this seminar was to bring together experts on computer algebra systems and to discuss various aspects of computer algebra software. The seminar covered the design of systems for differential equations, algebraic number theory, elliptic curves, term rewriting, algebraic geometry, invariant theory, representation theory, and symmetric groups. Furthermore, techniques for special problems in computer algebra like parallelization, building interfaces, garbage collection, memory management, and interoperability between systems were discussed.

The thirty eight participants came from seven countries. In addition to the official program, there was time for free discussions and informal meetings between participants. After the talks, the developers presented their systems and practical problems could be discussed. The nice atmosphere of the Dagstuhl Institute made this seminar a very enjoyable experience.

The organizers would like to thank everyone who contributed to the success of this seminar.

A B S T R A C T S

Σ^{IT} – A strongly-typed embeddable computer algebra library

Manuel Bronstein

We describe the new computer algebra library Σ^{IT} and its underlying design. The development of Σ^{IT} is motivated by the need to provide highly efficient implementations of key algorithms for linear ordinary differential and (q)-difference equations to scientific programmers and to computer algebra system users, regardless of the programming language or interactive system they use. As such, Σ^{IT} is not a computer algebra system per se, but a library (or substrate) which is designed to be "plugged" with minimal efforts into different types of client applications. The design characteristics of Σ^{IT} are:

- It is written completely in the $A^{\#}$ programming language, and has been designed from the ground up to take full advantage of the various $A^{\#}$ features: OO-category hierarchy, abstract data types, generic algorithms, etc...
- It is tuned for the manipulation of univariate skew-polynomials, rather than for the mathematical structures of commutative algebra. As a result, it provides efficient sub-algorithms for differential equations (currently symmetric powers and rational kernels).
- In addition to an `axiom`-like strongly-typed library of algorithms, Σ^{IT} provides an object-oriented expression tree type where the operators are themselves data types. Through various functions exported by the operators and through an included parser, this type is used for bidirectional communication with the "outside world" (TEX, C, `axiom`,

`maple`, etc. . .). A generic interpreter category permits the quick development of "Σ^{IT} -servers", i.e. typeless read-eval loops which give access to the various Σ^{IT} algorithms through process to process communication.

As a result, Σ^{IT} can be embedded into applications in two completely different fashions: either linked as a strongly-typed library into programs written in various compiled languages, thereby extending the type system by more than a hundred computer algebra data types, or used as a separate process with string-based interface from any interactive shell capable of reading and writing files. We demonstrate the latter use with a very short Maple program that computes LCLM-decompositions of completely reducible second order linear differential operators through the use of Σ^{IT} functionalities during that computation.

LiDIA – a C++ library for computational number theory

Johannes Buchmann, Thomas Papanikolaou

In our research group we implement the best known algorithms for factoring integers, factoring polynomials, discrete logarithms, linear algebra, lattice basis reduction, elliptic curves, permutation groups, and algebraic number fields. Furthermore, we make experiments with these algorithms and we try to improve them.

Highly efficient research software often uses assembler, is poor documented, has no source code available, is not well structured, and contains only little mathematical abstraction. Therefore, if algorithms should be improved rewriting seems often easier than reusing available software.

To avoid this waste of human work, we develop a system for computational number theory and computer algebra, called LiDIA, which tries to make the software written in our group

- verifiable

- reusable
- improvable
- extendable, also by others,

with **a minimal loss of efficiency**.

In this talk we report about this system LiDIA which consists of a very efficient C++ library, a small machine dependent kernel, and a user interface, called LC. The system is available from ftp@crypt1.cs.uni-sb.de.

Combining Term Rewriting and Computer Algebra in ReDuX

Reinhard Bündgen

We argue that term rewriting systems can profit from computer algebra systems and vice versa. Presenting the ReDuX term rewriting laboratory we show which of its modules benefit from modules of the SAC-2 computer algebra system. Then we describe in more detail how the reduction relation defined by a complete term rewriting system can be speed-up using built-in algebraic data types. We propose to use models of the rewrite specification as built-in data types that are freely generated by an (infinite) set of free generators. Such built-in data structures are called *evaluation domains*. A mapping that interprets a term to an object of the evaluation domain and then maps this object back to the normal form of the original term can then alternatively be used to compute normal forms. Most of the canonical rewrite specifications for algebraic varieties allow for evaluation domains that have a natural (i. e. efficient) evaluation function. The way the evaluation function works is independent of the intended model of the specification and it remains useful even if only a subspecification of a complete specification allows for an evaluation domain. In the latter case we speak of partial evaluation domains. ReDuX has been extended to support partial evaluation domains such that

rewrite specifications can be parameterized by built-in data types. Experiments with this extended version of ReDuX showed that partial evaluation domains can account for speed-ups of more than 1000.

Programming with Algebraic Structures and Morphisms: The Magma Language

John Cannon

The design of a Computer Algebra language is of necessity based on some particular view of mathematics. Analysis of systems such as Macsyma, Maple, Reduce and Mathematica show they are based on the idea of performing transformations on symbolic expressions belonging to a single fixed structure (usually some kind of differential ring). While this view may be appropriate for problems such as integration and the solution of differential equations, it is much less successful when used as the metaphor for computation in branches of mathematics such as algebra, number theory, geometry and combinatorics where the ideas of algebraic structure and structurepreserving transformation (morphism) are of fundamental importance.

A new model for the design of Computer Algebra systems based on the notions of algebraic structure (magma) and morphism has been devised. Magmas are first classified in terms of the algebraic variety to which they belong. The variety, of course, determines the operations and the axioms which these operations satisfy. However, to create a particular magma, we have to specify its (carrier) set and this is done through the notion of a category. For example, matrix rings, polynomial rings and power series rings are examples of (indexed) categories belonging to the variety of rings. Relationships between

magmas (e.g. A is a submagma of B, C is a quotient magma of D) are then naturally represented in terms of morphisms.

Magma is a new software system for algebra, number theory and geometry which has been designed in accordance with these principles. The use of the concept of a magma as the design basis provides a natural strong typing mechanism. Further, structures and their morphisms appear in the language as first class objects. Standard mathematical notions are used for the basic data types. The result is a powerful, clean language which deals with objects in a mathematically rigorous manner. An example creating the Galois correspondence between subfields and subgroups of a Galois extension of $\mathbb{Q}$ of degree 8 was presented as an illustration of the way in which magmas and morphisms provide the basis for a natural and concise means of specifying algebraic computations.

Parallel Long Integer Arithmetic

Giovanni Cesari

Multiple precision arithmetic forms the core of symbolic computation systems. A speedup of the basic algorithms can therefore improve all higher-level algorithms. Several packages have been developed for sequential processors. Although they were implemented carefully, they are still unsatisfactory for large inputs. Performance can be improved by choosing algorithms with better asymptotic behavior or by using other computing models such as parallel machines.

We have developed a parallel implementation of multiple precision arithmetic on distributed memory architecture. These algorithms will constitute the core of a parallel Computer Algebra library. We have seen that for large inputs parallel multiple precision arithmetic can offer very high performance improvements. Our target machine is the distributed memory Paragon. We have also studied how "portable" programs can be written using MPI (message passing interface) without sacrificing efficiency.

PARI / GP

Henri Cohen

PARI is a package designed for working very efficiently with mathematical types, especially those related to number theory, and contains a large number of sophisticated functions dealing with numerical analysis, elementary number theory and algebraic number theory. It can be accessed either as C / C++ library, or through an interpreter, called GP.

The inner kernel consists of a few hundred lines of assembly language code (one kernel per processor of course). Then comes a multiprecision kernel containing all operations on bigints and bigfloats.

In particular, the bigfloat implementation (which has been ported to other systems such as MAGMA) is mathematically much more correct then other such implementations found, e.g. in MMM.

Then a GEN (for generic) kernel contains programs for handling arithmetic operations between mathematical types.

Finally a large number of library packages (linear algebra, numerical analysis and transcendental functions, elementary and advanced number theory, plotting and input / output, ...) are available.

KANT – More than numbers

Mario Daberkow

The computer algebra system KASH for computational algebraic number theory has taken advantage of several software packages, that have not been developed by the KANT group, namely the C–kernel from the system Magma containing a memory management and basic arithmetic, the user interface from the computer algebra system GAP, the PVM–system for distributed computing and the database system mSQL.

We discussed several aspects of reusing developed software by showing the advantages and disadvantages of such a policy. For the system KANT it has been proven to be very successful to use existing software and the

overall effort to build interfaces between KANT the used software has been comparable moderate, e.g. to build the interfaces for the mentioned packages, at most 15% of the total source code was needed. To write these interfaces approximately 20% of the total programming time was necessary. Compared to the effort that would have been necessary to develop these modules alone, this is a rather moderate amount of source code and programming time. Finally, other important arguments for using these modules, are the facts that these libraries are already used for a long time and that they are still under development.

Computer Algebra Systems as Numeric Interfaces

J.H. Davenport

We describe work by Mike Dewar, Mike Richardson, Michael McGettrick, Godfrey Nolan, Ian Meikle, Bill Naylor, Brian Dupée and Simon Atkins. This team has achieved the following.

- An interface (called IRENA) between Reduce and the Nag Fortran Library — the library contained 700 routines, so full interfaces were developed to about 100 of them. This involved a great deal of work in developing uniform treatments of concepts like 'rectangles', which do not exist in Fortran, are natural to provide in a higher-level interface, and do not have a uniform parameterization in the Nag library.

- An interface between Axiom and the Nag library, available in Axiom 2.0 and beyond. This builds on the lessons learnt from IRENA, in particular the communications technology (now using a separate process, potentially on a remote machine) and the user interface (now a hypertext "fill in the boxes" interface of the same style as Axiom's "basic commands").

- A higher-level interface to `D03PSF`, a routine for solving hyperbolic PDEs, with 39 parameters, six of which are subroutines with a total of 59 parameters, and one of which is an array of 30 interacting options. Here we have developed both a generic interface (of the same style as above) and an interface to a specific PDE, Burger's equation $\frac{\partial u}{\partial t} - \frac{1}{2}\frac{\partial u^2}{\partial x} = 0$, which also incorporates calls to Axiom's graphical capabilities to display the result.

- An "expert system" ANNA, which takes a description of the mathematical problem (using the same style of interface) and decides which Nag routine would be most suitable for the problem, and then calls it via the interface already developed. Alternatively, the system may perform a symbolic transformation first. This system is intended for non-experts and for use in teaching.

- A "browser" for large symbolic structures such as matrices. This is designed to help with the situation where a symbolic system spits out a large number of lines for a single result, so that the user cannot see the wood for the trees.

REDLOG

Computer Algebra Meets Computer Logic
Parts I and II

Andreas Dolzmann and Thomas Sturm

REDLOG is a package that extends the computer algebra system REDUCE to a *computer logic system*, i.e., a system that provides algorithms for the symbolic manipulation of first-order formulas over some temporarily fixed language and theory. In contrast to general purpose theorem provers, the methods applied know about the underlying algebraic theory and make use of it. The focus is on simplification, parametric linear optimization, and quantifier elimination. We exhibit the REDLOG design issues and implementation techniques.

Why are Algebra Systems So Slow ?

John Fitch

This talk addresses the issue of software performance of algebra systems. For many years I have been concerned with creating such software, and have noticed that even acknowledged fast systems are capable of being improved. This is often the result of oversights by the designers or programmers, leading to inappropriate structures or algorithms.

After a survey of past system the main subject of the talk is a low level memory tracing tool which is provided in CSL, on which REDUCE is built, and preliminary results are presented which make explicit the ways in which memory is used when running a REDUCE application. This suggests a number of ways in which we can improve the CSL design, and also that of REDUCE itself.

The same mechanisms are also available in the related LISP system CCL. on which a future release of AXIOM will be based; and so the same techniques are expected to yield improvements in AXIOM as well.

The message is that performance engineering works.

SINGULAR

Gert-Martin Greuel, Hans Schönemann

The talk consisted of two parts: description of some of the principles and features of Singular and a report on an implementation of MP / MPP, a protocol to communicate polynomial structures between CAS.

Singular is a special system for Commutative Algebra, Algebraic Geometry and Singularity Theory. The basic algorithms include a very efficient standard (Gröbner) basis algorithm for general term orderings which need not well-orderings. This allows to compute and use Gröbner bases in general K-algebras as

- polynomial algebras, localization of these in prime ideals, tensor products of such algebras but also in non-commutative algebras as exterior algebras, Weyl-algebra.
- several ground fields K are available such as $\mathbb{Q}, \mathbb{F}_p$ (p=prime ≤ 32003), $\mathbb{F}_q$, $q = p^n$ small, algebraic and transcendental extensions.
- usual ideal theoretic operations and intersection, ideal quotient, saturation and more advanced algorithms using free resolutions of modules.
- combinatorial algorithms like dimension, multiplicity, Hilbert-series.
- algorithms for factorization uni- and multivariate polynomials.
- several libraries with procedures for primary decomposition, homological algebra, semiuniversal deformation etc., written in the Singular language (C-like).

Distributed computing and interoperability between specialized CAS lead to the problem of connecting these systems. We present a solution which is both general and efficient, based on the MultiProtocol MP.

Important points are the concepts of dictionaries, annotations and prototypes.

- Dictionaries define the semantics of operators, constants and annotations and therefore provide an easy way to extend the protocol.
- Annotations provide additional information about the data objects which may be very useful for the receiver, like the monomial ordering in a distributed polynomial.

- Prototypes are special annotations which describe the structure of the data (and can prove their homogeneity).

The encoding is not specific to a particular system; each MP-interface can parse the byte stream and, if it knows the dictionaries, understand it.

Generic symbolic programming in C++ – an example

Erich Kaltofen

Abstract: Given a vector space basis with integral domain coefficients, a variant of the Gram-Schmidt process produces an orthogonal basis using exact divisions, so that all arithmetic is within the integral domain. Zero-division is avoided by the assumption that in the domain a sum of squares of nonzero elements is always nonzero. In this talk we fully describe this method and use it to illustrate and compare a variety of means for implementing generic algorithms. Previous generic programming methods have been limited to one of compile-time, link-time, or run-time instantiation of type parameters, such as the integral domain of this algorithm, but we show how to express generic algorithms in C++ so that all three possibilities are available using a single source code. Finally, we take advantage of the genericness to test and time the algorithm using different arithmetics, including three huge-integer arithmetic packages.

Joint work with: Úlfar Erlingsson and David Musser (both RPI).

Computing Invariant Rings with the INVAR-System

Gregor Kemper

INVAR is a program package for invariant theory of finite groups. Consider the following setting:

Let K be a field and $G \leq \mathtt{GL_n(K)}$ a finite matrix group, which acts on the multivariate polynomial ring $R = K[x_1, \ldots, x_n]$ by linear substitutions of the variables. We are interested in the invariant ring $R^G = \{f \in R \mid \sigma(f) = f \ \forall \sigma \in G\}$. In particular, it is our goal to provide algorithms that calculate a finite system of generators of R^G as an algebra over K. This is done in two major steps:

1. Calculation of **primary invariants**. This is performed by a new algorithm that the author proposed with the aim of obtaining primary invariants of low degrees.

2. Calculation of **secondary invariants**. Here the two cases $\mathtt{char(K)} \mid |\mathtt{G}|$ ("modular") and $\mathtt{char(K)} \not| \, |\mathtt{G}|$ ("non-modular") split. In the non-modular case, there are linear algebra methods which use Molien's formula to obtain secondary invariants, whereas in the modular case a new algorithm using syzygy calculations and Gröbner bases is used.

The INVAR-system is written in the Maple programming language and is available in the Maple Share Library. A second version is in progress. The main computational requirements are Gröbner bases, primary decomposition and/or the Gröbner factorization algorithm, calculation of syzygy modules and linear algebra. Some of the tasks are performed by specialized systems such as SINGULAR or MACAULAY. Partial interfaces to these systems are included.

Gregor Kemper, The *Invar* Package for Calculating Rings of Invariant, IWR Preprint **93-34**, Heidelberg 1993.

Gregor Kemper, Calculating Invariant Rings of Finite Groups over Arbitrary Fields, J. of Symbolic Computation (to appear).

SYMMETRICA

Adalbert Kerber

The development of SYMMETRICA started 10 years ago with the diploma thesis of Axel Kohnert. The intention was to develop a computer algebra system for representation theory, combination theory, the theory of invariants of symmetric groups, and related classes of groups like the alternating groups, and the Kranz products of symmetric, alternating, and linear groups.

It should run on each computer using a C – compiler, should be written in standard C and should have an object oriented design (This was before the release of C++).

Over the years, a very large system has been developed, which is used intensively by many universities. Important parts of the system have been written by the group of A. Lascoux (Paris VII), the team of A. O. Morris (Aberystwyth), and the group of H. Lüneburg (Kaiserslautern).

The implemented data structures and algorithms for representation theory are based on multivariate polynomials. Important classes of these polynomials are the symmetric polynomials, in particular, the classical bases of the elemental and monomial symmetric polynomials, especially Schur-polynomials. There are also new generalizations available like Schubert- and zonal polynomials.

SYMMETRICA is public domain software and will be public domain in the future. The user of the system can obtain the source code including the LaTeX files of the handbook (which are unfortunately very incomplete) via the Internet.

One of the main differences of SYMMETRICA compared to other systems is the availability of procedures for computing irreducible, modular representations of matrices, projective ordinary matrix representations, decomposition numbers, and Brauer characters of symmetric groups.

New extensions of the system cover the constructive theory of discrete structures using finite group operations (We are also interested in Hecke-algebras at the moment).

As an example, we presented the application of the constructive theory of discrete structures, in particular the construction of 7 designs. Furthermore, we showed the connection to applications in chemistry (construction of all molecular graphs for a set of given data). Another application, which is under

development at the moment, is the computation of complete catalogues of irreducible linear codes.

Documentation of Algorithms and Proof of Results in Computer Algebra

Wolfram Koepf

We assume that a user who is not the developer uses a general purpose command (like `solve`, `integrate`, `sum`, etc.) in a general purpose system (like Axiom, Maple, Mathematica, Reduce, etc.).

If the system generates a result, then the questions arise: How can the user prove the result? How can the user receive information about the algorithms used? If no result is obtained, then: How can the system help the user to solve the problem?

We give examples of such situations, and propose the following:

- **Naming:** The algorithms should be called by their names
- **Citing:** They should be cited appropriately (e.g. in Help Page)
- **Variables:** Preferably the same variable names should be used as in the cited literature
- **Information Depth:** Rather detailed information about intermediate results should be accessible
- **Failure:** In case of failure an indication should be given which algorithms have been used, and which others might be applicable
- **Algorithms:** If available, algorithms should be used that enable mechanisms for fast proofs of their results

We give examples of an implementation of Gosper's and Zeilberger's algorithms in Maple which is on these lines. These algorithms bear rational certificates with which an easy proof of their results is possible.

PARSAC-2: Parallel Symbolic Computation on the desk-top

Wolfgang Küchlin

We give an introduction to programming methods and algorithms suitable for parallelizing Computer Algebra on modern multiprocessor workstations. Specifically we discuss multi-threaded programming and its use in the PARSAC-2 system for parallel symbolic computation. PARSAC-2 contains a system environment, called S-threads, which supports the multi-threading of symbolic algorithms in C for execution on a shared memory parallel machine. It also contains a distributed environment, called DTS, which can execute suitable threads of control remotely on the network. Finally we present our experience with some examples of parallel algorithms useful for solving systems of polynomial equations, including parallel methods for the computation of Gröbner Bases, multivariate polynomial GCD's, and root isolation of univariate polynomials. More detailed information may be found in:

W. Küchlin.
PARSAC-2: Parallel Computer Algebra on the Desk-Top.
In J. Fleischer, J. Grabmeier, F. Hehl, and W. Küchlin, editors,
Computer Algebra in Science and Engineering, pages 24–43,
Singapore, 1995. World Scientific.

An Invitation to an Open System for Computer Algebra Research

Rüdiger Loos

The first part summarizes the activities of the computer algebra research group at Tübingen. We concentrate on computer science aspects of computer algebra, in particular on language design for computing with algebra what we correlate with recent developments in functional programming languages like Theta (Liskov) or Sather-K (Goos). The second area is the definition of a portable interface for CA-libraries on the base of C++ and the Standard Template Library. These libraries should be useful for other research groups in CA through the internet.

The second part summarizes impressions from the Dagstuhl meeting. It was planned when it became obvious that at least four different CA research groups in Germany develop software independently, for example for computing Groebner bases. Compared with previous decades CA research has reached the working mathematicians and covers many aspects of current research activities in pure and applied mathematics, in chemistry, hardware verification, in physics and other applied sciences. We observed that the software technology used in CA is not anymore at the forefront of computer science. Many mathematicians use CS terminology in a vague and misleading manner, young programmers from mathematics reinvent methods and tools available for a long time in CS.

By these reasons in the third part we propose a common software research effort spanning many different groups both from mathematics and computer science. Primary goal is the reuse and interoperability of CA software across different groups, the focus on visible applications in other sciences and the cooperation with current CS research in deductive data bases, software specification and verification, reliable compiler construction, networking and interoperable libraries. New software developed in CA should even have a common language and common interface to other scientific libraries.

The computer algebra system CHEVIE and some applications

Frank Lübeck

CHEVIE is a joint project with: Meinolf Geck, Gerhard Wiš, Gunter Malle, Jean Michel, and Götz Pfeiffer.

CHEVIE is a special purpose computer algebra system for research mathematicians, interested in the representation theory of finite groups of the Lie type and associated structures, like Weyl-groups and Iwahori-Hecke-algebras.

The system consists of programs, written in the high level languages of the computer algebra system GAP and Maple, as well as library files readable by one of these systems.

The Maple-part of CHEVIE contains a library of generic ordinary character tables of series of groups of Lie type. They describe the character tables of sets of groups like $GL_2(q), SU_3(q), Sp_4(q), ...$, where q runs through all prime powers. Also, there is a library of tables of Green functions and there are programs to deal with these tables: look at values, print in TEX - format, show information on classes, characters or references to the literature, compute orthogonality relations, structure constants, tensor products (all calculations without specializing the q).

The GAP-part contains programs to compute in and with Coxeter groups and Iwahori-Hecke-algebras (programs for classical and library for (all) exceptional types).

The talk gave examples of how CHEVIE is developed in interaction with mathematical questions.

Furthermore, applications of the system in modular representation theory, Galois theory and group theory were mentioned.

The talk closed with a short overview on (almost existing) future extensions of CHEVIE: decomposition numbers of groups of Lie type and Iwahori-Hecke algebras; cyclotomic algebras; braid groups; a system of programs for computing (at least parts of) generic character tables (several single programs may be useful itself); new tables, in particular tables of unipotent characters of some groups up to rank 8.

Garbage Collection techniques for Computer Algebra

Gérard Milmeister

Garbage collection is an important issue in symbolic computation systems, especially interactive ones. There are several popular techniques: mark and sweep, stop and copy and reference counting. For all of them there exist variants that try to cope with their respective shortcomings.

In the AlgBench symbolic computation system we implemented a reference counting collector using smartpointers in C++. These simulate real (naked) pointers by wrapping them up in a hierarchy of classes parallel to the hierarchy of collected object classes. Smartpointers take care of the updating of reference counters.

We discuss the benefits and problems, especially those problems resulting from the limitations of the C++ compiler.

Mixfix Syntax for Computer Algebra

Stephan A. Missura

Mathematical notation is highly overloaded and strongly context dependent. Furthermore, it is extended incrementally by new operators and character-based parsing is needed. Hence, the corresponding grammar is usually ambiguous.

Mixfix notation is a very general method for defining new operator syntax and is therefore very well suited for supporting mathematical notation in interactive computer algebra systems. It supports prefix-, post-, infix-operators and also mixed forms. Juxtaposition and coercion operators fit very well in this scheme too.

Because the types of the declared operators and variables are used during parsing the parser already does type-checking and more strings can be parsed

in an unambiguous way than with conventional methods found in current computer algebra systems.

We discuss the design and implementation of a mixfix parser, especially the use of a higher-order, polymorphic type system and the new method of bracketing the token list produced by the scanner, which results in a reduction of the recursion depth during parsing.

The Design of a Computeralgebra-System

Holger Naundorf

When designing a general purpose computeralgebra-system, many different goals are set. The most important ones are generality and flexibility on the one hand and user-friendliness on the other hand.

Many people consider these goals as contradictory and hence try to find a satisfactory compromise. In contrast we consider generality and flexibility as a premise for user-friendliness, because every user prefers a different design. Hence the user must be able to change the system according to his/her ideas. Many examples of the necessity of flexibility can be given either for input, functionality and output.

Though a general and flexible system need not be user-friendly, it is usually possible to make it user-friendly — in most cases even for the user. The most general and flexible design is usually the easiest to implement due to the lack of special cases.

Due to these considerations we designed *MuPAD* emphasizing generality and flexibility. Relying on this we made *MuPAD* user-friendly.

Computing the class group of quadratic orders over principal ideal domains — a template implementation

Sachar Paulus

The most up-to-date algorithm for computing class groups of imaginary quadratic orders over the ring of rational integers was designed by Kevin McCurley after an idea of Martin Seysen. It is based on collecting relations by the reduction of randomly chosen binary quadratic forms. A generalization of this algorithm to quadratic orders over any other principal ideal domain requires either the existence of a reduction theory of binary quadratic forms with coefficients in this domain or another way of collecting relations. Both possibilities are examined for rings of integers of number fields resp. maximal subrings of congruence function fields and localizations thereof (whenever their class group is not trivial).

As a result, we conjecture that there exists a reduction theory of binary quadratic forms with coefficients in the maximal order of a totally real normeuclidean quadratic number field with totally negative discriminant. There is also a way of collecting relations similar to sieving techniques known from factoring which does not depend on reduction theory. A resulting generic algorithm has been implemented in C++ as TEMPLATE functions using LiDIA (for bigint and matrix computations), LEDA (for dictionaries of arbitrary types) and a TEMPLATE function package for finite abelian groups. Several examples are given; we compute e.g. a factor of the Jacobian of a hyperelliptic curve of type 1 over a finite field and show how the combination of sieving and reduction in the classical cases yields major improvements in practice.

The KANT Project

Michael Pohst

KANT is a software package specializing in sophisticated computations with algebraic number fields. The more elementary tools used - like memory management, basic arithmetic, finite fields, linear algebra, and lattice theory - are adopted from Magma. The calculations in number fields, however, are based on algorithms which were either developed or at least improved by members of the KANT group. In this talk we report on recent progress with relative extensions of number fields. Major achievements are an algorithm for detecting subfields and the potential of computing Hilbert class fields. The user can access the KANT routines via the KANT-Shell KASH. It was developed by using GAP and allows Maple like programming. KANT is public domain and is available from: ftp.math.tu-berlin.de at pub/algebra/Kant/Kash.

How To Lift a Library

Sibylle Schupp

Generic algebraic libraries run the risk of losing efficiency compared to non-generic libraries. To solve the seeming antagonism between genericity and efficiency, we propose a technique which we call *lifting*. The lifting process is an abstraction process which starts with a well-known efficient algorithm and abstracts carefully from it. As lifting guarantees that fast non-generic algorithms instantiate their generic lifts, we preserve computing time. Apart from generic programming, lifting is interesting for computer algebra in conceptual and conceptional respects.

Interoperability between Computer Algebra Systems

Gábor A. Simon

In the development of a computer algebra algorithm one would frequently like to use (perhaps only temporarily) existing algorithms, which are implemented in different, possibly incompatible systems. This means, that one needs possibilities which allow the cooperation and/or communication between programs, applications, objects and environments despite differences in the implementation language, execution environment and model abstraction. All these possibilities are meant by the notion of interoperability . The basic aspects of interoperability are: (i) the control aspect, caring about the coordination of the inter-operating programs, (ii) the data aspect, establishing type compatibility for the shared data objects and (iii) choosing transport services and low-level communication protocols achieving an efficient transmission of the data. There are several groups working on such problems in the computer algebra (OpenMath, Posso XDR, Central Control, ASAP, MP, CAS/Pi, MathLink, MathEdge etc.) Most of the research activities in this field concentrate on the data aspect (or type model), looking for a single universal representation, either on a character or byte stream basis. The implementations are usually based on low-l evel transport mechanisms, which are optimized for performance, rather than for the ease of programming, reliability, flexibility and extensibility. For an application programmer it is difficult to manage these very technical aspects concerning the transport mechanism, even in connection with his/her own system and much more with foreign systems.

Distributed object managers, as SOM/DSOM, OLE, CORBA etc. try to provide a convenient infrastructure for the development of distributed applications. In this paper, we summarize our experiences with a prototype implementation for a communication interface between SAC–2 and LiDIA, using the CORBA like Inter–Language Unification (ILU) system from Xerox PARC.

SAC–2 is a traditional CA system, using its own garbage collection and the implementation language C. LiDIA is a recent one implemented in C ++. We will show, what components would be need e. g. to use the LLL

(Lenstra, A.K., Lenstra H.W., Lovász, L.) algorithm reducing lattice bases implemented in LiDIA, from a SAC–2 application written in ALDES. For the demonstration sake we will here examine a simple example, computing the determinant of a matrix with arbitrary precision integer coefficients.

The method we present helps to reduce the complexity of developing distributed applications to solve common problems in the computer algebra. The developer do not need detailed knowledge about the foreign system, to set up a server application. Users of the client application need not to care about localizing a server application, it will be found automatically using the support of the infrastructure provided by the ILU system.

Fast Multiprecision Arithmetic The Turing Processor System

Ekkehart Vetter

TP, developed by A. Schönhage (`schoe@cs.uni-bonn.de`), is a collection of about 200 efficient routines for long integers (including Schönhage-Strassen multiplication), fast gcd computations, and for large scale computations with real and complex numbers and polynomials, all within arbitrary and guaranteed precision. The routines, although written in a special, assembler-like programming language TPAL, are compiled into C and can be used within nearly any environment. The underlying hypothetical machine bases on a Turing machine and a RISC processor. Its tapes—each cell storing a whole 32-bit word—turn out to be well suited for the efficient implementation of computer arithmetic algorithms.

This hypothetical machine can be efficiently emulated on real machines. Special care is taken of the implementation of the self-growing tapes. Here one important aspect is locality: The local tape access, characterizing Turing machines, circumvents the memory bottleneck of current computer architectures. The tape management API allows an efficient communication and parameter passing from and to TP.

An optimizing compiler translates TPAL into C (optionally mixed with assembler statements). Thus, the emulation is quite portable. Summarizing, beside an environment for the design and in-depth evaluation of computer arithmetic algorithms, TP provides a fast multi-precision kernel for computer algebra and numerical applications. More information about TP can be obtained from : `ftp://ftp.cs.uni-bonn.de/pub/tp/HTML/TP.html`.

Theorem Proving in Cancellative Abelian Monoids

Uwe Waldmann

A theorem prover that is to be used for applications such as program verification and synthesis has to cope with the fact that theories from which program properties are to be derived are divided into several parts: some specifying standard mathematical structures, including numbers, lists, multisets, graphs, others providing the axioms for additional free function and predicate symbols. The latter describe, in a more or less ad hoc manner, objects, and their properties, of the particular domain of application for which a program is to be written. The prover must thus combine mathematical with meta-mathematical reasoning; it must be able to deal with unknown domains, but also to deal *efficiently* with known domains.

Traditional theorem proving techniques like resolution are ill-suited for mathematical reasoning, as they do neither recognize nor make use of existing algebraic structure. Even worse, axioms like associativity, commutativity, or transitivity allow an enormous number of inferences and produce large numbers of equivalent variants of given formulae. To overcome these problems it is necessary to integrate the algebraic axioms as inference rules into the general proving calculus. We describe a refined superposition calculus for cancellative abelian monoids. Our calculus requires neither explicit inferences with the theory clauses for cancellative abelian monoids nor extended equations or clauses. Improved ordering constraints allow us to restrict to

inferences that involve the maximal term of the maximal sum in the maximal literal. Furthermore, the search space is reduced drastically by certain variable elimination techniques [1].

References:

[1] Harald Ganzinger and Uwe Waldmann. Theorem proving in cancellative abelian monoids. Technical Report MPI-I-96-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany, January 1996.

Programming Language Support for Memory Management

Stephen Watt

We look at the problem of memory management and implications in computer algebra. It is observed that the biggest gains do not come from improving the speed of allocation / deallocation / garbage collection, but rather from avoiding memory allocation to begin with. The code transformation methods of the $A^{\#}$ compiler are presented, with emphasis on procedural integration (cross-file inlining) and data-structure elimination. Finally, we make a case that programming language support (source language semantics) offers significant potential to improve memory behaviour. We propose that by limiting write access to objects to an explicitly declared initial part of their lifetime that we obtain the flexibility to in-place updating operations to create values and then the generational memory and the transformable / optimizable code asfeets of read only objects. Language support can enforce no. aliasing of objects during the initial, writable part of their life without imposing an unbearable burden for the usual case where many read-only mathematical objects are handled.

Computer Algebra Software for Algebraic Geometry (CASA)

Franz Winkler

The program package CASA (Computer Algebra Software for constructive Algebraic geometry) is designed for performing computations and reasoning about geometric objects in classical algebraic geometry, in particular affine and projective algebraic geometry over an algebraically closed field F of characteristic 0. Moreover, the field F has to be a computable field in the sense of the underlying computer algebra system Maple, i.e. all the arithmetic operations have to be available in the system. In the absence of any special indication of the field, CASA always assumes that the field of computation is the rational numbers $\mathbb{Q}$ or a finite algebraic extension thereof.

CASA has been developed at RISC-Linz over the last years, starting in 1990, by a research group under the direction of the author. It is mainly developed as a result of various ph.d. and masters theses done in the computer algebra group at RISC-Linz. Consequently, the functionality provided strongly reflects the research interests of the people working on the development of CASA. The current version of the system, CASA 2.2, is built on top of the Maple computer algebra system, in particular Maple V.3.

The objects that CASA deals with are algebraic sets in four different representations.

- *Implicit representation:* An algebraic set is the set of common zeros of a system of polynomial equations. To give an algebraic set in implicit form means to give finitely many polynomials.

- *Projected representation:* As a consequence of the primitive element theorem every irreducible d–dimensional algebraic set in n–dimensional space is, after a suitable linear transformation of coordinates, birationally projectable onto an irreducible d–dimensional algebraic set in $(d+1)$–dimensional space, which can be specified by a single polynomial in $d+1$ variables. This can be generalized to unmixed-dimensional algebraic sets. An algebraic set in projected form is given by a polynomial and a tuple of rational functions (specifying the birational mapping).

- *Parametric representation:* Some irreducible algebraic sets can be parameterized by rational functions. An algebraic set in parametric form is given by a tuple of rational functions that parameterizes the algebraic set.

- *Representation by places:* All algebraic curves can be parameterized by a set of power series that are convergent around a point of the curve. An algebraic set is given by places if for each branch passing through a certain point on the algebraic set a tuple of power series that parameterizes the algebraic set around the point is specified.

CASA also works with the polynomial ideals corresponding to these geometric objects.

The operations available in CASA include

- ideal theoretic operations $+, *, \cap, /$,
- creating algebraic sets in different representations,
- generating curves of fixed multiplicities at given points,
- intersection, union, and difference of algebraic sets,
- computing tangent cones and tangent spaces,
- computation of the dimension of an algebraic set,
- decomposition into irreducible components,
- transformations of algebraic sets to hypersurfaces,
- computation of the genus of a curve,
- rational parameterization of curves,
- implicitization of parametrically given algebraic sets,
- Puiseux series expansions,
- plotting both explicitly and implicitly given curves and surfaces.

An early version of the system was described in [1]. For more information on the underlying mathematics and software issues we refer to [3]. Some typical sessions of CASA may be found in [4]. A reference manual of the current version can be found in the technical report [2]. CASA is available at the ftp server `ftp.risc.uni-linz.ac.at` in the directory `/pub/CASA`.

References:

- [1] R. Gebauer, M. Kalkbrener, B. Wall, F. Winkler, *CASA: A Computer Algebra Package for Constructive Algebraic Geometry*, in: Proc. ISSAC'91, 403–410, S.M. Watt (ed.), ACM Press (1991).
- [2] M. Mňuk, B. Wall, F. Winkler, *CASA Reference Manual (Version 2.2)*, RISC Report 95-05 (1995).
- [3] B. Wall, *Symbolic Computation with Algebraic Sets*, Ph.D. Dissertation, RISC (1993).
- [4] F. Winkler, *Constructive Algebraic Geometry with CASA*, talk at the workshop CoCoA, Cortona (1993) and RISC Report 93-26 (1993).

Display, Generation, and Evaluation of Mathematical Structures

Tom Wickham – Jones

This talk will examine mathematical notation and how it can be used in computer mathematics systems. It will consider the development of modern notation and its relation to mathematics and to computer systems. The display of notation by classical printing and the WWW will then be discussed, followed by consideration of the methods by which these display forms can be generated. It will be shown how the most natural methods lead directly to a powerful integration of live interactive mathematics structures in computer mathematics systems.

SIMATH - a computer algebra system with an emphasis on elliptic curves

Horst G. Zimmer

SIMATH is a computer algebra system written in C. A list system serves as the basis for all SIMATH types such as integers, rationals, matrices and vectors. In addition to the source libraries containing all SIMATH functions the system is equipped with the programmable calculator *simcalc* by which most functions can be handled in an interactive mode.

Emphasis lies on algorithms for elliptic curves. Fundamental procedures concern

- The determination of torsion groups of a large class of elliptic curves over number fields of degree ≤ 4.
- The computation of the rank and a basis of the Mordell-Weil-group over $\mathbb{Q}$ and certain quadratic number fields with and - if possible - without assuming the Birch and Swinnerton-Dyer conjectures.
- The determination of all integral and S-integral points of elliptic curves over $\mathbb{Q}$.
- The construction of elliptic curves over large finite fields with group of rational points of given order - with application to
  - the search for large prime numbers
  - the determination of elliptic curves which are of cryptographical relevance.

Reporter: Markus Maurer (Saarbrücken)