Summary of New Features in Magma V2.6

November 8, 1999

1 Introduction

This document provides a terse summary of the new features installed in Magma for release version V2.6 (November 8, 1999). Previous releases of Magma are: V2.5 (July 7, 1999), V2.4 (December 3, 1998), V2.3 (January 30, 1998), V2.20 (April 18, 1997), V2.10 (October 14, 1996), V2.01 (June 21, 1996), and V1.30 (March 5, 1996); release notes for these versions are found in the document `relprev.dvi`.

2 Summary

- The Magma machinery for permutation groups has been re-implemented from scratch using new data structures and in many cases, new or improved algorithms. The new data structures allow many algorithms to be implemented more efficiently. Further optimizations based on the new data structures will be introduced in subsequent releases.

- Among the many new algorithms introduced for permutation groups, new polynomial-time algorithms are used for finding various critical normal subgroups such as an elementary abelian regular normal subgroup (of a primitive group), the soluble radical, the p-core and many others.

- Advanced algorithms for the computation of greatest common divisors (GCD) and extended greatest common divisors (XGCD) for integers have been implemented.

- The module for p-adic rings and fields have been rewritten to fix problems encountered with the existing one (inherited from Pari). The functionality has been extended and it is intended that it will be extended further, especially with the introduction of local rings (see below) for which p-adic rings are the basis.

- The first stage of a major facility for working with completions of local rings and and their fields of fractions is released in Magma V2.5.

- The module for working with Newton polygons has been re-implemented and extended.

- Algorithms due to Walker and Duval for computing Puiseux expansions of the roots of a bivariate polynomial over a field have been implemented.

- The machinery for finding cliques in a graph has been extended: it is now possible to distinguish between cliques as complete graphs and maximal cliques.
- The machinery for finding resolutions and parallelisms in incidence structures has been extended.

3 Groups

3.1 Permutation Groups [HB 26]

A complete re-implementation of permutation groups has been undertaken. This has given us the opportunity to revise all permutation group algorithms in Magma. All these algorithms have been updated, with many improvements made, and several entirely new methods implemented.

New Features:

- The **AbelianNormalQuotient** function uses work of Easdown and Praeger to construct the quotient by an abelian normal subgroup as a permutation group of degree not greater than that of the original group.
- A non-trivial abelian normal subgroup may be located using the **AbelianNormalSubgroup** function.
- **CentralizerOfNormalSubgroup** implements a polynomial time algorithm to find the centralizer of a normal subgroup. This algorithm is now used for evaluating the **Centre** function.
- The maximal subgroups of many groups can be found using the **MaximalSubgroups** function.
- The **FrattiniSubgroup** function has been extended to all groups for which maximal subgroups can be found.
- Quotients of permutation groups corresponding to certain natural G-actions may be computed using the **TransitiveQuotient** and **PrimitiveQuotient** functions.
- The function **SocleQuotient** computes a compact permutation representation of G over its socle when G is primitive or a trivial Fitting group.
- A process to step through a right transversal for a subgroup is available through the **TransversalProcess** function. With this process there is no need to store the transversal, so it can be applied to subgroups of very large index.
- The function **BurnsideMatrix** returns the Burnside matrix of a group.

New Algorithms:

- The **NormalSubgroups** and **NormalLattice** functions now use the algorithm of Cannon and Souvignier. The result of NormalLattice no longer contains conjugacy class information about the subgroups.
- A new algorithm, based on homomorphic reduction, is employed to implement the function **MaximalNormalSubgroup** – thereby replacing the former method which was based on the use of conjugacy classes.
- **Normalizers** are now found using a fusion of the backtrack search methods of Butler and of Holt.
- **IsAffine** and **EARNS** are now implemented using polynomial-time algorithms rather than by backtrack searching.
- bf SolvableRadical, **pCore**, **ElementaryAbelianSeries** and related solvable and abelian normal subgroup functions are now computed using the (polynomial time) algorithm of Unger. This avoids finding Sylow subgroups of the group.

– The lifting algorithm as employed by function **ConjugacyClasses** has been improved using the affine reductions of Mecky and Neubüser (option **Extend**).

– **RandomSchreier** and similar algorithms now routinely limit Schreier tree depth to give probable short Schreier vector base and strong generating sets.

– Many changes have been made to homomorphism calculations that have improved their speed dramatically.

– The strategy employed in the **Verify** function to determine the switch-over point from STCS to BCS methods, and improved its choice of base points for the BCS method.

– Whenever possible standard group constructions (AGL, PGL, PGO, DirectProduct, etc) now assert the order of the constructed group to facilitate later construction of a base and strong generating set.

Removals and Changes:

– The function call **GSet(G, S)**, where G is a permutation group and S is an indexed set, has been changed to give the (derived) action of the group on the elements of the set rather than on the indices of the elements, which was the case previously. To obtain the action of a permutation group on the indices of an indexed set the new function **GSetFromIndexed(G, S)**, should be used.

– The function **pCoreKantor** has been withdrawn.

– The functions **Strip(G, x)** and **WordStrip** now return three values: First a boolean value reporting "x in G", next the residue of the strip and last the level of the residue.

3.2 General Matrix Groups [HB 27]

Bug fixes:

– **Centralizer(G, H)** has ben changed so that H is no longer required to be a subgroup of G.

4 Rings

4.1 Integer Ring [HB 31]

Advanced asymptotically-fast algorithms for the computation of greatest common divisors (GCD) and extended greatest common divisors (XGCD) have been implemented. Hitherto, Magma has only contained the classical Accelerated GCD algorithm of Weber (which itself is very fast for small and medium inputs) and the classical Euclidean XGCD algorithm.

The fast classical Lehmer XGCD algorithm has been implemented, which itself is about 5 times faster than the Euclidean XGCD algorithm. Also, the Schönhage recursive ("half-GCD") algorithm has been implemented, yielding asymptotically-fast GCD and XGCD algorithms.

On a Sun Ultrasparc workstation, the crossover point for the Schönhage GCD algorithm (when it beats the classical Accelerated GCD algorithm) is 32768 bits (about 10000 decimal digits), while the crossover point for the Schönhage XGCD algorithm (when it beats the Lehmer XGCD algorithm) is 6000 bits (about 2000 decimal digits).

On a 200Mhz Sun Sparc workstation, Magma V2.6 can compute the GCD of two arbitrary integers, each having a million decimal digits, in 16.6 seconds, and the extended GCD multipliers for the same numbers in 30.6 seconds. (The previous algorithms in Magma took 2273 and 41935 seconds respectively for the same computations!)

On the same machine, Magma V2.6 can compute the GCD (which is 1 of course!) of the consecutive Fibonnaci numbers F_N and F_{N-1} for $N = 10^8$ [sic], where each number has about 20 million decimal digits, in 3.8 hours, and the extended GCD multipliers for the same numbers in 6.4 hours! (Consecutive Fibonacci numbers provide worst-case input for GCD algorithms.)

New features:

- New fast Lehmer (classical) extended greatest common divisor algorithm.
- New asymptotically fast Schönhage recursive GCD algorithm for large integers.
- New asymptotically fast Schönhage recursive XGCD algorithm for large integers.
- Improvement of the rational reconstruction algorithm, as a consequence of the above new algorithms.

4.2 p-adic Rings and Fields

The module implementing p-adic rings and fields has been rewritten to overcome problems encountered with the version inherited from Pari. The functionality has been extended and it is intended that it will be extended further, especially with the growth of the local rings (see below) for which the p-adic rings are a base.

Removals and Changes:

- The function `^` has been removed where both arguments are p-adic elements.
- The behaviour of the `Log` function has changed. For input not of the form $1+x$ where x has positive valuation `Log` returns an error since the series expansion of the logarithm function will not converge.
- The function `Truncate`, which has the same functionality as `Expand`, will be removed in a future release and so is not documented.

New Features:

- The `SeriesPrinting` attribute allows the user to specify the print style of a p-adic element.
- `Prime` returns the prime number that has valuation 1 in the p-adic ring or field.
- `ResidueClassField` returns the quotient of the p-adic ring by its unique maximal ideal. In the case of fields, the residue class field is defined to be that of its ring of integers.
- `UniformizingElement` returns the prime of the p-adic ring or field as an element of that ring or field.
- `Precision` returns the precision of the ring or field. This is used as the default precision to create elements of that ring or field with.
- `ChangePrecision` allows the user to change the precision of a ring or field or an element in either.
- The function `Random` has been provided to generate random elements in a p-adic ring.
- `GreatestCommonDivisor` is now available for polynomials over fixed precision p-adic rings.
- The function `NewtonPolygon` and the associated machinery can be used to compute the valuations of the roots of a polynomial over a p-adic ring and the number of roots that have that valuation.
- Hensel Lifting of approximate roots of polynomials and approximate polynomial factorizations over a p-adic ring is now provided by the function `HenselLift`.
- The functions `IsSquare` and `IsPower` have been added as companions to `SquareRoot` and `Root`.

Bug Fixes:

- Speed and Memory problems have been addressed resulting in the possibility of certain operations such as large matrix multiplications that were previously impossible.
- Precision can be easily retrieved for a p-adic structure using the function `Precision`.
- Echelonization has now been implemented for p-adic rings in addition to being available for fields.

4.3 Local Rings and Fields

The first stage of a major facility for working with completions of local rings and and their fields of fractions is included in Magma V2.6.

New Features:

- To create a local ring or field or a p-adic structure the functions `LocalRing`, `LocalField`, `pAdicRing` and `pAdicField` have been provided.
- A local or p-adic ring or field can be extended using the functions `UnramifiedExtension` and `TotallyRamifiedExtension`.
- The ring of integers of a local field can be found using `IntegerRing` and the field of fractions of a local ring can be found using the function `FieldOfFractions`.
- The `SeriesPrinting` attribute can be used to print local elements as series in the uniformizing element or as polynomials over the subring.
- `Prime` returns the prime number with valuation 1 in the p-adic substructure.
- The degrees of the unramified part and the totally ramified part of the ring and field can be retrieved using `InertiaDegree` and `RamificationDegree` respectively and the total degree by `TotalDegree`.
- `DefiningPolynomial`, `EisensteinPolynomial` and `InertialPolynomial` return the polynomials that define the local ring or field as an extension of the p-adic or inertial substructures.
- `InertiaRing` and `InertiaField` return the unramified part of the local ring or field. Similarly, `PrimeRing` and `PrimeField` return the p-adic substructure.
- `ResidueClassField` computes the quotient of the valuation ring by its unique maximal ideal.
- `UniformizingElement` returns the element of valuation 1 and `InertialElement` returns the element adjoined in making the unramified part of the extension.
- `ChangePrecision` allows the precision of a ring or field or an element thereof to be altered.
- `Random` and `Representative` have both been provided for local rings.
- The `elt<>` constructor is provided to create elements of a ring or field.
- Imprecise zero elements can be created using the function `BigO`.
- The sequence conversions given by `LocseqInert`, `InertseqPadic`, `Locseq` and `ElementToSequence` provide several different ways to represent an element of a local ring or field as a sequence.
- Arithmetic and Predicates are provided as usual.
- `Parent` returns the local ring or field that an element is contained in.
- The precisions of a ring or field element can be retrieved using `Precision`, `AbsolutePrecision` and `RelativePrecision`
- To consider an element to have full precision apply `Expand`.
- The valuation of an element is returned by `Valuation`.
- The logarithm of an element is computed by `Log` and the exponential by `Exp`.
- The norm and trace of a local element over the p-adic substructure can be calculated using `Norm` and `Trace`.

– The greatest common divisor of two polynomials over a local ring can be calculated using the `GreatestCommonDivisor` function.

– The valuations of the roots of a polynomial over a local ring and the number of roots with each valuation can be determined using `NewtonPolygon` and the associated machinery.

– Approximate roots of polynomials over local rings and approximate factorizations of polynomials over local rings can be improved using `HenselLift`.

– Whether a polynomial has a root in a local ring can be determined by using `HasRoot`.

– The functions `Root`, `IsPower`, `SquareRoot` and `IsSquare` find n-th roots or square roots of elements in local rings and fields.

4.4 Puiseux Expansions and Newton Polygons

Newton Polygons have been rewritten and some applications added for Magma V2.6. A small portion of the original Newton polygon machinery has been removed while functions using Newton polygons have been added.

Removals and Changes:

– The equality functions have been removed due to concerns over the definitions of equality. These include `eq`, `CompactlyEqual`, `LowerEqual` and `OuterEqual`.

– `IsDegenerate` has been removed.

New Features:

– Walker's algorithm (R.J. Walker 1978) for computing Puiseux expansions has been implemented in `PuiseuxExpansion` with further expansion of the series beyond the precision first specified being allowed by `ExpandToPrecision`. `ExpandToPrecision` uses the implicit function theorem which has been implemented in `ImplicitFunction`.

– `IsPartialRoot` and `IsUniquePartialRoot` use the implicit function theorem to determine whether a series is the beginning of an expansion of a root of a polynomial over a series ring and further in the latter case whether this expansion is the beginning of one or more roots.

– `PuiseuxExponents` returns the first exponents of the terms of a series up till the reduced denominator of the exponent is the exponent denominator for the series. `PuiseuxExponentsCommon` determines the exponents of the leading terms of two series expansions which are equal.

– In some cases Puiseux Expansions can also be calculated using `DuvalPuiseuxExpansion`. This algorithm (D. Duval 1989) is faster than Walker's in most cases but the polynomials for which it works are less numerous. It returns expansions in the form of parametrizations. `ParametrizationToPuiseux` and `PuiseuxToParametrization` have been provided to convert between the two forms of information.

– The functions `Roots` and `HasRoot` have been added for polynomials over series rings. In addition to the general polynomial function `Roots` allows an argument specifying the precision to which the roots should be given.

5 Incidence Structures

5.1 Graphs [HB 68]

The clique functionality has been revised so that whenever possible, the user may differentiate between cliques which are maximal and those which are not. Further, maximal independent sets or independent sets of a given size k in a graph G can be easily found by finding maximal cliques or cliques of size k in the complement of G. For this reason, only two functions which are concerned with independent sets are now provided: one finds a maximum independent set and the other returns the independence number of a graph. *Note that cliques and independent set functions only apply to undirected graphs.*

Changes:

- The function `HasClique(`G, k`)` returns true if and only if the graph has a clique of size k. A third argument permits the user to test for the existence of a clique or of a maximal clique. A fourth argument a permits the userm to test for the existence of a maximal clique of size greater than or less than k. A parameter allows one to choose which algorithm is to be used.
- The function `MaximumClique` finds a maximum clique in the graph. It is possible to specify which algorithm is to be used. There is a dual function `MaximumIndependentSet`, which finds a maximum independent set.
- The function `AllCliques(`G`)` finds all maximal cliques in G.
- The function `AllCliques(`G, k`)` finds all maximal cliques of size k in G. This functions admits a third argument which allows one to find all –not necessarily maximal– cliques of size k. In the later case, an additional parameter allows one to choose which algorithm is to be used.
- The functions `Clique` and `AllIndependentSets` are no longer documented in the Magma handbook. They will be removed from the next version of Magma.

5.2 Incidence Structures and Designs [HB 69]

The functions dealing with resolutions of a design have been renamed and new functionality has been provided.

Changes and new features:

- The function `HasResolution(`D`)` returns the value true if and only if the incidence structure D has a resolution. If true, one resolution is returned as the second value and the index of the resolution is returned as the third value.
- The function `HasResolution(`D, λ`)` returns the value true if and only if the incidence structure D has a resolution with index λ.
- The function `AllResolutions(`D`)` returns all resolutions of the incidence structure D.
- The function `AllResolutions(`D, λ`)` returns all resolutions of the incidence structure D with index λ.
- The function `IsResolution(`D, P`)` tests if the collection P of blocks (or sets) is a resolution of the incidence structure D. If true it also returns the index of the resolution.

– The function `HasParallelism(`D`)` tests if the uniform incidence structure D has a parallelism. If true a parallelism is returned as the second value of the function. A parameter allows one to specify the algorithm to be used.

– The function `AllParallelisms(`D`)` returns all parallelisms of the uniform incidence structure D.

– The function `IsParallelism(`D, P`)` tests if the collection P of blocks (or sets) is a parallelism of the incidence structure D.

– The function `HasParallelClass(`D`)` is true if and only if the uniform incidence structure D has a parallel class. If this is the case, one such class is returned as the second value.

– The function `AllParallelClasses(`D`)` returns all parallel classes of the uniform incidence structure D.