An object oriented concept for elliptic curves

by Marc Conrad and Susanne Schmitt

Abstract

We describe objects and methods for an object oriented concept of elliptic curves. In order to achieve this, the representation of elliptic curves in the computer algebra systems LiDIA and SIMATH is analyzed. This leads to the objects *models* (objects which realize Weierstraß equations) and *elliptic curves* (objects which realize the part of elliptic curves which is invariant under isomorphism). The distinction between methods defined over arbitrary fields and methods defined over concrete fields involves the idea of a polymorphic field. We consider elliptic curves over finite fields and over number fields, but the concept may also be extended for elliptic curves over any other field.

1 Introduction

In the last years computations with elliptic curves have been a main field of research. One reason is the increasing importance of elliptic curves in cryptography (see e.g. [1], [6], [2]). But there has also been progress in number theory itself, as the computation of rank or torsion points of an elliptic curve, or the search for integral points (e.g. [10], [4], [5], [9]). The aim of this paper is to develop a concept which proposes the way elliptic curves should be implemented in software.

In [7] the *demands* on software for computations in number theory in the context of a computer algebra system are formulated (efficiency, reuseability, adaptability, extensibility, portability, simplicity and reliability). We give two examples how these demands relate to computations with elliptic curves. A detailed discussion is given in Section 4.

- The demand for *reuseability* implies the possibility to reuse code written for elliptic curves over one field (e.g. $\mathbf{F}_p$) also for other fields (e.g. $\mathbf{Q}$). So we need a clear distinction between algorithms available for arbitrary fields (e.g. the computation of the j-invariant) and algorithms which depend on the basic field (e.g. the counting of points on the elliptic curve over $\mathbf{F}_p$).
- *Simplicity* implies that the design has to follow the mathematical intuition (e.g. elliptic curves may be interpreted as a class of isomorphic equations).

Only an object oriented approach as it is presented here can follow these demands. Our approach to provide a full design document before starting the implementation is still unusual in mathematics although well established in other areas. The advantage is obvious as the software can be implemented in various contexts, e.g. an extension to an existing Computer Algebra System, a stand alone programm, a Java package, or a C++ library.

Although basically we are mapping mathematical objects to objects in software design, the task of finding an appropriate design is nontrivial: A mathematical textbook definition of an elliptic curve does not a priori explain the attributes, methods and associated objects of an object "elliptic curve". Also the existing software packages for elliptic curves do not implement a comprehensive concept. However, in order to get a better understanding which methods and data structures are appropriate we first give in Section 2 a short overview on existing implementations of elliptic curves. Especially, we have a closer look at the implementation of elliptic curves in two packages which are designed as general purpose systems, namely LiDIA and SIMATH.

Using the ideas of Section 2 we develop in Section 3 object hierarchies concerning elliptic curves and describe the methods of the involved objects. As the subject is approached from the mathematical point of view, the concept is developed in a general context of an object oriented environment without specifying to a certain existing programming language, as C++, Java, or Smalltalk. For a comprehensive description of such an object oriented concept independent of a specific programming language see [3]. A general discussion on the implementability of different object oriented concepts in a variety of languages can also be found in [3]. In addition we discuss in Section 5 implementation issues as far as they are related to the proposed concept for elliptic curves here.

Based on the structures for elliptic curves presented in Section 3 we discuss in Section 4 possible extensions and other advantages of the concept.

We assume in the following that the reader is familiar with the arithmetic of elliptic curves. For details see e.g. [8].

2 The representation of elliptic curves in LiDIA and SIMATH

We focus in this section on approaches to realize object oriented ideas for elliptic curves in existing software. It is not the aim of this paper to given a comprehensive overview on software for elliptic curves or to describe in detail the functionalities of the distinct packages. For this, we refer to the manual pages of these packages.

Software which deals with elliptic curves can be divided into two classes. First there are packages which are designed for special applications as e. g. *mwrank* of J. Cremona for the computation of the rank of elliptic curves over the rationals or *ellcalc* of N. Bruin which is designed for the investigation of a special class of elliptic curves over number fields and finite fields. The implementations of these packages are not intended for the use in a general context, and therefore none of these packages realizes a general concept for the representation of elliptic curves. However, they contain useful ideas concerning object oriented representation. For instance *mwrank* combines data for elliptic curves in C++ classes, and *ellcalc* contains – in a similar way as LiDIA – code which is used both for finite fields and number fields.

Second, there are software packages which are intended for a broad spectrum of applications. Aside of the commercial system Magma where the source code is not publicly available, and Apecs, which is based on Maple, there are two general purpose systems in number theory, namely LiDIA and SIMATH, which aim to provide elliptic curves in a highly structured way.

The computer algebra system LiDIA interprets an elliptic curve as an equation in Weierstraß normal form over a field which can be $\mathbf{Q}$, $\mathbf{F}_p$ where p is prime, or $\mathbf{F}_{2^n}$. The implementation language of LiDIA is C++. In order to realize the polymorphic behaviour, i. e. doing computations for different fields, a template class is used to represent the elliptic curve. The functionality of this class includes the computation of the Tate values, discriminant, and j-invariant. The defining equation can also be modified by giving an explicit birational transformation. A special flag which indicates if the equation is already in short Weierstraß normal form allows to increase the performance in the calculations as there are special routines called when this flag is set.

In addition to the basic arithmetic over arbitrary fields, there exist several functions valid only for elliptic curves defined over finite fields. These are for example the computation of the group order, the isomorphism type or a test on supersingularity. In order to use these methods a special variable must be defined during the compilation process, which makes the handling unwieldy. However, the distinction between field dependent and field independent methods is clearly a reasonable approach.

The elliptic curve is initialized with the values a_1, a_2, a_3, a_4, a_6 representing the equation $y^2 + a_1y + a_3xy = x^3 + a_2x^2 + a_4x + a_6$. There is no distinction between the elliptic curve itself and a model of the elliptic curve. This can – from the mathematical point of view – be confusing, as e.g. Tate values are not properties of an elliptic curve but only of a model of the curve.

The computer algebra system SIMATH is, as a C library, a priori not object oriented. It contains algorithms for elliptic curves over the rationals, number fields, and finite fields and gives therefore a quite comprehensive overview about existing algorithms. However, the implementation is not field independent as in LiDIA.

For SIMATH, we focus here only on the implementation of elliptic curves over quadratic number fields and the field of rationals as it provides a concept which can be used to develop object oriented structures. The implementation of elliptic curves over number fields of higher degree and over finite fields does not follow such a concept. For a comprehensive description of the algorithms used in SIMATH see for example [10] or the manual pages.

SIMATH takes into account that different models are needed to perform special algorithms. For example a model in short Weierstraß normal form with integral coefficients is required for the computation of the torsion group of an elliptic curve over the rational numbers, using the criterion of Nagell, Lutz, and Cassels. Minimal models are required for many higher algorithms, as e.g. the computation of the reduction type, the L-series of the curve, or the computation of the conductor. Elliptic curves over $\mathbf{Q}$ and over quadratic number fields are implemented in SIMATH in such a way that the user has the possibility to access these models directly.

We describe now the elliptic curve over a quadratic number field in more detail (the concept for elliptic curves over the rational field is similar). The elliptic curve is represented as a list $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4, \mathcal{L}_5)$. The list $\mathcal{L}_1$ represents the actual model of the curve, i.e. the Weierstraß equation which was originally given to define the curve. The list $\mathcal{L}_2$ represents a global minimal model for the curve, if such a model exists. If it does not exist, it contains lists which represent the local minimal models at the prime ideals of bad reduction. The list $\mathcal{L}_3$ represents a model of the curve in short Weierstraß normal form with integral coefficients.

All these three lists contain the data which depends on the model, as there are the coefficients $a_1, \ldots, a_6$ of the model, the Tate values, the discriminant, norm and factorization of the discriminant. Also contained in these lists are special points of the equation as torsion points or generators of the Mordell-Weil group, and birational transformations to the other models.

The list $\mathcal{L}_4$ contains constants which are independent of the model. Examples of such constants are the j-invariant, the conductor of the curve, the rank and the structure of the torsion group. The last list $\mathcal{L}_5$ contains information about the quadratic number field.

An elliptic curve over a quadratic number field in SIMATH is initialized with the values $a_1, \ldots, a_6$ which are stored in $\mathcal{L}_1$. The other lists are empty after the initialization. When some of the data is needed the first time it is computed and stored at the right place. When the data is needed again, it is simply read out.

The description of LiDIA and SIMATH above shows two important principles for the design of an object oriented concept.

(1) In order to work with different models of the same curve simultaneously one has to distinguish clearly between the elliptic curve and models of the elliptic curve. Here, SIMATH provides at least three important models.

(2) In the context of elliptic curves there exist functionalities which depend strongly on the base field and functionalities which can be performed in any field. LiDIA approaches this idea by implementing elliptic curves as a template class.

3 The object oriented representation of elliptic curves

For the object oriented representation of an elliptic curve we follow the two principles mentioned in Section 2 and translate them in object oriented terminology.

(1) An elliptic curve over an arbitrary field is a plane curve of genus one with a rational point. This curve can be given in different isomorphic equations. In this context there are data which depend on the equation and data

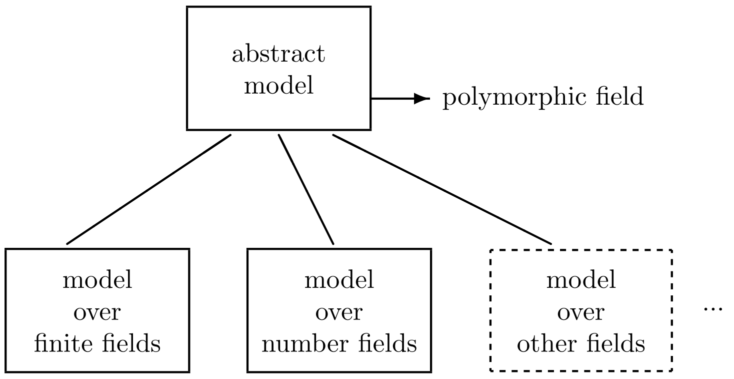

Figure 1: Object hierarchy of models of an elliptic curve.

which are invariant under isomorphisms. In an object oriented environment the idea of having data leads to methods which are bound to certain objects. As objects we introduce here *models* with methods dependent on a specific equation and *elliptic curves* with methods which are invariant under birational transformations.

(2) Many basic algorithms (e.g. the computation of Tate values, discriminant, or j-invariant) can be performed for models or elliptic curves over arbitrary fields. For this we use a polymorphic implementation of a field. For a polymorphic field we may assume that all standard field operations ($+$, $-$, $\cdot$, $\dots$) are available. In an application the polymorphic mechanism finds the appropriate realization of operations depending on the concrete field during the execution. The idea of operating in a polymorphic field leads to the distinction between abstract objects and concrete specifications.

We describe now the objects *model* and *elliptic curve* on the abstract level. After that we show how these abstract objects should be extended (by inheritance) to objects over number fields and finite fields. Of course, this concept can be extended to implement elliptic curves also over other fields (see also Section 4). Figures 1 and 2 give an overview of these objects. At the end of this section we sketch the object oriented representation of points according to the concept developed for elliptic curves.

We assume that an elliptic curve is given by a Weierstraß normal form (but see also Section 4).

3.1 The abstract situation

The abstract model has the following features:

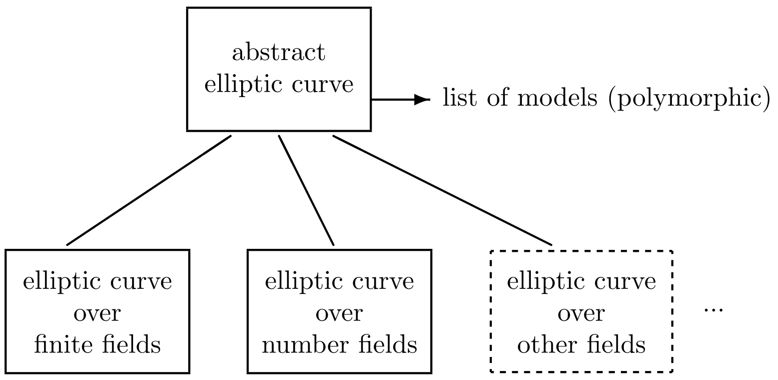
abstract elliptic curve
list of models (polymorphic)
elliptic curve over finite fields
elliptic curve over number fields
elliptic curve over other fields
...

Figure 2: Object hierarchy of elliptic curves.

(1) It is constructed by the values a_1, a_2, a_3, a_4, a_6, representing the equation $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ or by giving a birational transformation from another existing model.

(2) It contains a reference to the elliptic curve which it belongs to. So the model is always in the context of an elliptic curve. This can be achieved in two ways:

a) If the construction of the model is called by an elliptic curve the reference is set to this curve.

b) Otherwise the model constructs an elliptic curve, putting itself in the list of models as the "actual model".

(3) It has methods for the computation of the Tate values and the discriminant. Note that the j-invariant is independent of the model and is computed by a method of the elliptic curve.

(4) It can check if a point lies on the model.

(5) It provides the addition of two points and the multiplication with $n \in \mathbf{Z}$ of a point on the model.

(6) It has a reference to a polymorphic field.

The methods described in (1) to (5) can be implemented in the abstract model using the reference to the polymorphic field. Of course, this does not exclude the possibility to override certain methods in a concrete specification (see Section 4).

Note that the implementation of (2b) contradicts the idea of type checking at compile time, as the information about which concrete elliptic curve should be constructed is only available at run time. This causes difficulties in statically typed languages as C++. Dynamically typed languages, as Smalltalk, overcome this problem, but have other disadvantages. For a discussion on different implementation languages see [3] and Section 5.

An elliptic curve can be considered as an isomorphism class of models. So, the object *abstract elliptic curve* contains a list of polymorphic references to different, isomorphic models. Here, 'list' stands for any appropriate aggregate type as e.g. arrays or mappings.

A typical problem in the context of elliptic curves is "Let E be the elliptic curve given by the equation $y^2 = x^3 + 3x + 4$ over $\mathbf{Q}$. Compute a basis of the group of rational points". So, questions about elliptic curves are often related to a specific model. Therefore we stress one of the models as the *actual model*: the model which has been used to define the curve.

The abstract elliptic curve has the following features:

(1) It is constructed by a model which is then the actual model.

(2) It has facilities to manage the list of models. That means to add a model if it is isomorphic to the actual model, to return a specified model etc. It must also be possible to clearly distinguish the different models, e.g. by giving them names.

(3) It provides for two models a birational transformation between these models.

(4) It has a method to compute the j-invariant.

(5) It has a method to analyze the structure of the group of rational points. We note that this method would be too difficult (perhaps impossible) to be implemented independent from the field where the curve is defined. It is therefore recommended to implement the method as a deferred method (i.e. a method which must be overridden in the specifications). Because the *question* for the structure of the group of rational points is common to all fields, we think that this method should not be omitted on the abstract level.

(6) It has a convenient and simple mechanism to call the methods of a model, e.g. the computation of the Tate values, the addition of points, etc. This can be implemented by defining as methods of the elliptic curve the methods of the model with an optional additional parameter. This additional parameter is the model where the method is actually performed. If the parameter is not given the actual model is chosen.

3.2 Elliptic curves over number fields

A model over number fields is derived from the abstract model and in addition has the following features:

(1) It has methods to compute the torsion points and generators of the torsion group.

(2) It provides the computation of generators of the group of rational points.

(3) It has methods to find the integral or the S-integral points for a finite set of places S.

(4) It computes an estimate for the difference of the Weil height and the Néron-Tate height.

Note that today general implementations of these algorithms do not exist. However, there are theoretical algorithms for elliptic curves over arbitrary number fields, which can realize the methods (1) to (4). Because these methods describe canonical questions for elliptic curves over number fields, we suggest to return a value "unknown" in cases where there is at present no implementation available.

An elliptic curve over number fields is derived from the abstract elliptic curve and in addition has the following features:

(1) It provides a model in short Weierstraß normal form.

(2) It provides a global minimal model if it exists, otherwise a local minimal model to every place P of bad reduction, and the reduction type of the curve at P.

(3) It computes the conductor of the curve.

(4) It has methods to compute the sign of the functional equation, the real and the complex period, the rank r, the value of the r-th derivative of the L-series at $s = 1$, and the order of the Tate-Shafarevich group. It should be considered in the implementation that today there are only algorithms available which give lower or upper bounds for the rank, or which use the conjectures of Birch and Swinnerton-Dyer for computing the rank or the order of the Tate-Shafarevich group.

(5) It computes the regulator of the curve.

3.3 Elliptic curves over finite fields

A model over a finite field is derived from the abstract model and in addition has the following features:

(1) It has methods which compute special points of the curve e.g. points of large order or generators of the group.

(2) It has a method to solve to a given point and an integer the discrete logarithm problem.

An elliptic curve over a finite field is derived from the abstract elliptic curve and in addition has the following features:

(1) It provides a model in short Weierstraß normal form, if it exists. If there is no such model (which can happen in characteristic 2 or 3) it provides some other well defined normal form.

(2) It computes the Hasse interval.

(3) It has a method to compute the number of points of the group.

(4) It computes the Hasse invariant in order to check if the curve is supersingular.

3.4 Points on elliptic curves

The hierarchic concept for points on elliptic curves is similar to the concept described in the previous subsections. We distinguish between *abstract points* (points defined over an abstract field) and points defined over concrete fields, derived from the abstract points.

The abstract point has the following features:

(1) It can be constructed in different representations (especially affine, Jacobian, and projective coordinates) and the representation can be changed during computation.

(2) It has a polymorphic reference to the model it belongs to. If the point happens to appear not in the context of a model, the reference is not defined.

(3) It can test if it is the point at infinity.

(4) An abstract point has also a method to compute its order, which is overridden in the concrete specification. This is realized analogously to the method for the analyzing of the structure of the rational points of an abstract elliptic curve, as described above.

The concrete specification of a point over number fields has in addition a method to compute its naive height and its canonical height.

Note that from the mathematical intuition the addition of points belongs to the model and therefore it is a method of the abstract model.

4 The concept in view of software engineering

Object oriented design is a well established method for designing software and provides a lot of advantages. For further information in a general, non mathematical context see e.g. [3]. We focus here on these issues which are related to the elliptic curve concept and follow the categories given in [7]. Note also that some of the following categories overlap (e.g. simplicity increases the efficiency in developping and extensibility implies reuseability).

Efficiency. An object oriented program has usually more overhead than (for instance) a pure C program or assembler code. This will be more than compensated by the increase in efficiency in maintaining and using the system and by the advantages concerning reuseability, extensibility, adaptability and simplicity described below. See also Section 5 for a discussion on decreasing runtime overhead.

Reuseability. The code written in the abstract model and the abstract elliptic curve can obviously be used without modification for other fields (rational and finite function fields or complex numbers) and even for commutative rings using an exception mechanism to catch invalid division. Thus, an elliptic curve integer factorization can be easily obtained by specifying a modular integer class as a subclass of the abstract field throwing an exception when a division cannot be performed. When the exception is caught by the program a factor of the modulus has been found.

Adaptability. Mathematical knowledge is developing as well as knowledge in writing good algorithms. We give some examples showing that the concept is flexible enough to reflect future improvements.

- There is currently no implementation for the computation of the Mordell-Weil group in an arbitrary number field while for some fields (e.g. quadratic number fields) an implementation exists. Our concept proposes to implement a method "unknown" as a possible return value. Improving and implementing appropriate algorithms reduces the number of "unknown" cases but does not require a redesign of the classes.
- The concept is independent of a specific integer arithmetic. So the system can adapt easier to other arithmetics.
- One of the features of the abstract elliptic curve class is that it manages a set of models, connected each other by birational transformations. For current algorithms, as they can be found e.g. in SIMATH only a small number of models is necessary for a computation. Therefore in a first stage the model handling could be implemented in a simple and straightforward way, e.g. by using a language provided container class.

  In the future it might be important for an application to work on a large number of isomorphic models at the same time. It might then be necessary to implement the models in a graph structure where the nodes are models and the edges are birational transformations between two models. Mapping a point from one model to another would then require to find a path through this graph. The design proposed here makes no explicit statement of the nature of the model handling mechanism, but defines only the expected behaviour of the elliptic curve class. So the design has not to be changed when the internal implementation is updated.

Extensibility. The concept in Section 3 gives the basic structure for elliptic curves. In an object oriented environment it is possible to add new objects or new methods to existing objects (if the source is available, otherwise classes have to be extended using inheritance). We sketch here some examples for such extensions.

- We focus in this article on algebraic number fields and finite fields. However, the abstract model and elliptic curve can (and should) also be used to define models and elliptic curves over other fields. For instance an implementation over rational function fields opens the way for computations with parametrized classes of elliptic curves.
- It is recommended to have special objects derived from the abstract model and elliptic curve with additional and overridden methods for special fields. For instance, because of the increasing importance in cryptography, there should be specialized methods for models/elliptic curves over $\mathbf{F}_{2^n}$ and $\mathbf{F}_p$ where p is a (large) prime. These methods implement basic routines in a highly specialized way to obtain efficient code. Our concept allows to encapsulate these methods in a class which is derived from the model over finite fields.
- We also do not make in Section 3 extra considerations for the field $\mathbf{Q}$ because it can be seen as a special case of an algebraic number field. However, depending on the intended application, elliptic curves and/or models over $\mathbf{Q}$ may be implemented as objects on their own, possibly derived from the objects over algebraic number fields.
- For some applications it is vital to use other models than models in Weierstraß normal form. Although theoretically there is always a transformation into a Weierstraß normal form it can be practically impossible to find this transformation. For such a case the concept can be extended by introducing a *generalized model*, which contains at least the defining polynomial equation and a method to transform this equation to a Weierstraß normal form. This method then should be allowed to have *unknown* as return value in order to make it possible to check at runtime if a Weierstraß normal form can be computed. The object *elliptic curve* should also have the possibility to be constructed by such a model.
- In LiDIA there is a special flag indicating that the defining equation is given in short Weierstraß normal form because many computations can then be performed faster. In the object oriented concept as it is proposed here this would be achieved by deriving an object *model in short Weierstraß normal form* from the object *model*. The increase of performance will then be realized by overriding appropriate methods of the model.

Reliability. Object oriented software usually provides more reliable programs [7]. Concerning elliptic curves this is important for algorithms where it is difficult or impossible to verify the result e.g. the computation of the structure of the point group. Also for the application of elliptic curves in cryptography reliability is crucial.

Simplicity. Being able to implement an object, i.e. an entity which has data and behavior, we are able to map one to one "mathematical objects" which are defined by data (e.g. $a_1, \ldots, a_6$) and are related to operations (as point addition) into software. So, the design is close to the mathematical intuition. This is especially true for "abstract" mathematical objects as an elliptic curve over an arbitrary field.

5 Implementation issues

In the previous sections we intended to discuss the concept from a point of view which is independent from a specific programming language. In this section, we finally discuss some aspects in a concrete environment. We focus on C++ and Java. Other languages (Smalltalk, Object C, Delphi, ...) are uncommon in mathematical context and the performance problems are similar as they will be discussed below for C++ and Java.

A crucial point in the concept lies in the use of structures defined on an abstract level (abstract elliptic curve, abstract model, polymorphic field). In C++ abstract classes are usually implemented via virtual methods. In Java this is supported by default, i.e. by not declaring methods as final.

In both languages the overhead arises by finding at run time appropriate methods in a given class hierarchy. In the following, we will first discuss several aspects before proposing a strategy based on currently available software.

- The curve and model hierarchies designed here are very flat. Both for the model and the elliptic curve hierarchy there is only one abstract base class serving as parent class. A virtual table lookup should therefore be less time consuming than in average code, especially when the compiler is able to identify flat hierarchies and optimize code appropriately.

- The model and curve could be implemented as templates (with the known disadvantages of code duplicating etc). Unfortunately polymorphic behavior is difficult to handle with templates, as the template implementation of elliptic curves in LiDIA shows: A special flag has to be set at compile time to use some features of elliptic curves over finite fields. However, template mechanisms are to be refined and improved in future C++-compilers and they should therefore be kept in mind as a serious alternative for implementing this design.

- Another approach to reduce runtime overhead is the use of *interfaces* (provided in Java) instead of abstract classes as interfaces are more restrictive than abstract classes. Methods previously implemented on an abstract level have then to be implemented on the concrete level. This is acceptable for the field hierarchy, but it seems not reasonable for the curve and model hierarchies, as many methods there are intended to be implemented on the abstract level.

Another important point, aside of the problem concerning virtual methods, is the need for an efficient underlying basic (integer) arithmetic. A lot of computation time will be spend here as many application areas of elliptic curves (as cryptography or diophantine equations) require the use of long integers, that means integers which are larger then 32 or 64 bit. An efficient arithmetic as e.g. the gnu gmp package (a C library) is therefore necessary, and there is few hope to achieve similar performance with a Java package.

From the above we propose the following approach which is a compromise between a pure object oriented implementation in Java classes and (runtime) efficiency issues. The abstract model class should be linked to the underlying field hierarchy via a Java style interface. The arithmetic of each field has to be implemented with the help of a fast C/C++ library. The model and elliptic curve classes should be implemented as Java classes in inheritance hierarchies, closely following the design proposed here. Compiling Java code into executable code while linking it to a C/C++ library is for instance supported by the gcj compiler. An experimental implementation is currently a project by the authors.

Finally it should be noted that we hope that our design (maybe together with similar contributions for other mathematical areas) will also influence the development of compilers and/or programming languages towards the special demands of mathematicians. At least, explicit development of designs is a necessary condition for this.

References

[1] K. Araki, T. Satoh, S. Miura (1998). Overview of elliptic curve cryptography. *Public key cryptography. PKC '98, Pacifico Yokohama, Japan*, *Lect. Notes Comput. Sci.* **1431**, 29–49.

[2] I. Blake, G. Seroussi, N. Smart (1999). *Elliptic Curves in Cryptography.* Cambridge University Press.

[3] T. Budd (1998). *An Introduction to Object-Oriented Programming.* Addison-Wesley.

[4] J. E. Cremona (1997). *Algorithms for modular elliptic curves.* Cambridge University Press.

[5] J. Gebel, A. Pethő, H. G. Zimmer (1997). Computing integral points on Mordell's elliptic curves. *Collect. Math.* **48**, 115–136.

[6] N. Koblitz (1998). *Algebraic Aspects of Cryptography.* Springer.

[7] Th. Papanikolaou (1997). *Entwurf und Entwicklung einer objektorientierten Bibliothek für algorithmische Zahlentheorie.* Dissertation an der Universität des Saarlandes, Saarbrücken.

[8] J. Silverman (1986). *The arithmetic of elliptic curves*. Springer.

[9] N. P. Smart, N. M. Stephens (1997). Integral points on elliptic curves over number fields, *Math. Proc. Camb. Philos. Soc.* **122.1**, 9–16.

[10] H. G. Zimmer (1996). Basic algorithms for elliptic curves. In *Number theory (Eger 1996)*. 541–595, de Gruyter.

References to Software

[11] Apecs. *Arithmetic of plane elliptic curves*, ftp://ftp.math.mcgill.ca/pub/apecs.

[12] N. Bruin. http://www.math.uu.nl/people/bruin/.

[13] gcj. *The GNU Compiler for the Java Programming Language*, http://gcc.gnu.org/java.

[14] gmp. *A library for arbitrary precision arithmetic*, http://www.gnu.org/software/gmp/gmp.html.

[15] LiDIA. *A C++ library for computational number theory*, http://www.informatik.tu-darmstadt.de/TI/LiDIA.

[16] Magma. *A Computational Algebra System for Algebra, Number Theory and Geometry* http://www.maths.usyd.edu.au:8000/u/magma.

[17] mwrank. *A package to compute ranks of elliptic curves over the rationals*, http://www.maths.nott.ac.uk/personal/jec/ftp/progs.

[18] SIMATH. *A computer algebra system for algorithmic number theory*, http://www.simath.info.