# Sage Enhancement Proposal: Global Number Theory

David Roe

June 12, 2007

## 1 Introduction

Currently Sage supports absolute and relative number fields, and fractional ideals. However, in some ways relative extensions in Sage still behave like the corresponding absolute extension (in printing, for example). There are also many structures related to number fields, such as orders, ideals and fractional ideals, class groups, unit groups, galois groups, completions, adeles and ideles that remain either unimplemented or poorly integrated into Sage. This SEP presents a plan to expand the scope of global number theory native to Sage.

There are a number of existing programs that have facilities for these kinds of computations, including Pari, Magma, KANT, and GiNAC. Because only Pari is open source, Sage currently uses Pari for most of the difficult computations associated with number fields. Yet all of these systems have lessons to teach us about useful structures and interfaces. In particular, Magma's interface provides much of the inspiration for the improvements suggested in this SEP.

We first overview the classes implementing number fields, orders, ideals and fractional ideals, class groups, unit groups, Galois groups, completions, adeles and ideles.

### 1.1 Overview

There should be a variety of classes representing number fields in Sage, one for each kind of data that can define a number field. For example, there will be a class for number fields defined by an irreducible polynomial over $\mathbb{Q}$,

a class for number fields defined by an irreducible polynomial over another number field $K$, a class for a number field defined by another number field $L/K$ and a subgroup $H$ of $\mathrm{Gal}(L/K)$, as well as others. Some of the different kinds of number fields will require different types of elements, while others won't support elements. For more details, see section 2.

Number theorists frequently need to work with rings of integers in number fields. More generally, Magma supports orders in number fields; Sage should do the same. All orders will be sandwiched between an equation order and the corresponding maximal order. How to implement orders is a little bit ambiguous, since they are simultaneously rings, $\mathbb{Z}$-modules, and in the case of a relative extension $L/K$, $\mathcal{O}_K$-modules. This question of representation is complicated by the fact that not all orders have power bases, and when $\mathcal{O}_K$ is not a principal ideal domain, they are not always free $\mathcal{O}_K$ modules. The implementation of elements of orders also raises some questions: does one represent elements as $\mathbb{Z}$-linear combinations of a $\mathbb{Z}$ basis for the order, as $\mathcal{O}_K$-linear combinations of an $\mathcal{O}_K$ pseudo-basis, or as $K$ linear combinations of a power basis of $L/K$. See section 3 for more information.

In addition to elements of number fields and orders, we also want to be able to represent ideals. Again, we have multiple choices for representation. For an ideal $I$ of an order $\mathfrak{O}$ over a maximal order $\mathcal{O}_K$, we can give generators for $I$ as an $\mathfrak{O}$ module, generators as as $\mathcal{O}_K$ module or a basis as a $\mathbb{Z}$ module. We also want to be able to store fractional ideals, and there are similar choices there. See section 4.

Given a number field $L/K$, we have a natural homomorphism of groups from $L^\times$ to the group of fractional ideals of $L$. The kernel of this map is the unit group of $L$, and the cokernel is the class group. We want to be able to represent these not only as abstract groups, but also with the maps that fit them into the four term exact sequence: given an element of the unit group, we should be able to get an element of $L^\times$ that it maps to, and given a fractional ideal, we should be able to find it's class in the class group. See section 5.

Likewise, we want to be able to work with the Galois group $\mathrm{Gal}(L/K)$ not just as an abstract group, but also with its natural action on $L$. This action should fit into a larger framework of group actions in Sage, which needs to be developed. See section 6.

A number field $L/K$ and a place $\mathfrak{w}$ of $L$ over $\mathfrak{v}$ of $K$ determine a completion $L_\mathfrak{w}$ over $K_\mathfrak{v}$. Sage should support such completions, whether they be at archimedean or non-archimedean , as well as appropriate automatic coercion

of elements. In addition, sage should have support for adèles and idèles. See section 7.

# 2 Number Fields

One reason to have different types of number fields for different types of defining data is the conservation of computational time and space. A number field K and a set of places of K define a maximal abelian extension unramified outside those places. Even though this data uniquely defines a number field, some quantities associated to this number field are more difficult than others to compute. For example, the defining polynomial of the number field is much more difficult to compute than the degree or the discriminant. By having a structure that defines a number field using class field theory data, the computation of the defining polynomial is postponed until it is actually needed.

The more important reason to have a variety of different kinds of number fields is a question of representation. Even though the number field defined by $x^8 - x^4 + 1$ is isomorphic to that defined by adjoining $\sqrt{2}$, $\sqrt{3}$ and $\sqrt{-1}$, at times one is more suitable and at times the other. Similarly, it is important to be able to work with relative extensions: all computations should be relative to the given ground field, and elements should be written as polynomials with coefficients in the ground field.

All number fields will have `NumberField_generic` as a superclass. This class will implement the common functionality shared by all of the number fields, It will also provide a unified interface for functions supported by number fields and for conversion between different types of number fields.

Even though the rational field does derive from `NumberField_generic`, mathematically it is a number field. Since Magma is more strongly typed than Sage, they do not consider $\mathbb{Q}$ a number field. But Sage should: by having the classes `RationalField` and `Rational` implement all of the appropriate functions, it is quite reasonable to have `is_NumberField` return true on input the rational field.

On the parent level, `NumberField_generic` will have subclasses `NumberField_absolute`, `NumberField_relative`, `NumberField_multiple`, `NumberField_subfield`, `NumberField_fixed_fi` and `NumberField_abelian_extension`.

The class `NumberField_absolute` will support absolute number fields, defined by a single irreducible polynomial over $\mathbb{Q}$. I'm not totally convinced

that there's a good reason to separate this class from relative extensions that happen to have $\mathbb{Q}$ as the ground field (which are still permitted), but there there are certainly some facts true about absolute extensions that are not true in general (no unramified extensions of $\mathbb{Q}$, etc. Thoughts here would be appreciated.

The class `NumberField_relative` will support relative number fields, defined by a single irreducible polynomial over another number field $K$. This class will be the workhorse of the number field classes. The base field $K$ must be one of `RationalField`, `NumberField_absolute`, `NumberField_relative` or `NumberField_multiple`, though these choices may expand in the future.

The class `NumberField_multiple` will allow extension of a number field $K$ by adjoining the roots of polynomials $f_1, \ldots, f_m$. The polynomial $f_{i+1}$ must be irreducible over $K[x_1, \ldots, x_i]/(f_1(x_1), \ldots f_i(x_i))$. The ground field is $K$, and the user provides names for the roots of the $f_i$. With this representation, certain computations may be easier in some cases: Galois groups for example. In addition, elements can be represented in terms of user given set of generators, rather than always in terms of a primitive element.

The class `NumberField_subfield` will allow specification of a number field as a subfield of an existing extension $L/K$ by giving elements $\alpha_1, \ldots, \alpha_m$ of $L$ in order to generate the smallest subfield of $L$ containing $K$ and all of the $\alpha_i$. With $m = 1$, this type of number field will be very similar to `NumberField_relative`; for $m > 1$ it will be very similar to `NumberField_multiple`, though there should not be a condition that the $\alpha_i$ are independent as required for `NumberField_multiple`... However, `NumberField_subfield` comes with a specified embedding into $L$.

The class `NumberField_fixed_field` will allow creating number fields from subgroups of Galois groups. Given a Galois extension $L/K$ (will this also work for non-normal extensions?) and a subgroup $H$ of $\mathrm{Gal}(L/K)$, create the fixed subfield of $L$ associated to $H$. As with `NumberField_subfield`, this type of number field has a specified embedding into $L$. In fact, elements of either of these could just be implemented as elements of $L$ with a different parent.

Finally, the class `NumberField_abelian_extension` will allow creating number fields using some sorts of class field theory data. I'm not sure exactly how this should work, besides just working directly off what Magma does.

`NumberField_absolute` and `NumberField_relative` can have the same class of elements. `NumberField_multiple` will need to have a different class, though both types should presumably inherit from a single `NumberFieldElement_generic`

class. The two types of subfields don't need their own types of elements. Finally, I can't think of a way to even represent elements of an instance of `NumberField_abelian_extension` that lie outside the ground field without knowing the defining polynomial. I think it's reasonable to require a conversion to another type of number field before allowing creation of elements.

# 3 Orders in Number Fields

Support for orders in Sage should be similar to that in Magma. In particular, one needs to be able to extend orders as one extends number fields (by adjoining a root of a polynomial) and also by adding new integral elements in the same number field. The first process should be done by calling the `extend/extension` function on another order. Magma has a restriction that one can only create extensions of maximal orders; we should think about why they made this choice. I haven't come up with a good name for the second process yet.

Orders should generally be stored as modules over their ground order. There is a theory of modules over Dedekind domains (this is probably why Magma requires that the ground order be a maximal order): basically one stores a pseudobasis and a fractional ideal of the ground order for each basis element. One then has the option of checking whether a given pseudobasis and corresponding set of fractional ideals defines a ring. In the case that the ground order has class number one, we might want to change representation, since in this case the order in fact is a free module. In particular, orders over $\mathbb{Z}$ will be free $\mathbb{Z}$ modules.

Thus as with number fields, we should have corresponding different types of orders. It's not clear to me how it would be useful to have an `NumberFieldOrder_abelian_extens` class, but all of the other corresponding classes, `NumberFieldOrder_absolute`, `NumberFieldOrder_relative`, `NumberFieldOrder_multiple`, `NumberFieldOrder_subfield` and `NumberFieldOrder_fixed_field` make sense.

One point about orders on which I question Magma's choice is what to do with the fraction field of an order. Magma gives this object it's own type, which keeps track of the order that it came from. Elements of this fraction field are represented differently from elements of the corresponding number field. Is the benefit of this alternative representation for elements and keeping track of the originating order worth the extra type of structure? Perhaps.

# 4 Ideals

Given an order, one needs to be able to create both normal and fractional ideals. As I mentioned in the overview, there are a number of choices for default representation. I would like to get some more discussion on this point.

# 5 Unit and Class Groups

The various types of number fields and orders should support the computation of unit and class groups. These should certainly be in a form so that casting in appropriate direction works (though perhaps not automatic coercion?). We need to figure out the best way to do this while still using Pari for the back end (at least in the short term).

# 6 Galois and Automorphism Groups

I want to get computation of Galois and automorphism groups working so that one can use them naturally with elements. They should have natural actions on appropriate fields, and inertia and decomposition groups should also exist. One should be able to construct fixed fields associated to subgroups.

There should also be G-modules so that we can eventually implement Galois cohomology.

# 7 Completions, Adèles and Idèles

There should be a natural way to complete any number field or order at a prime ideal or infinite place and get the appropriate local object. There should be automatic coercion supported with elements of the resulting completions.

Adèles and Idèles should exist. Perhaps one needs to specify them at the set of places where they behave badly (and all infinite places), and provide a function that can generate an appropriate local element given a place. One might also need to specify a norm for the element as a whole... I'm not sure.