p-ADIC RINGS AND FIELDS: A PROPOSAL FOR **SAGE**

This is a record of a conversation which occurred on 6 October 2006 concerning the desired structure for p-adic rings and fields, and their extensions. Members include Iftikhar Burhanuddin (USC), Kiran Kedlaya (MIT), David Roe (Harvard), John Voight (Minnesota), and Joe Wetherell (San Diego).

Throughout, p is prime! And this is just a draft, many of these ideas are shooting from the hip.

1. p-ADIC RINGS

For $\mathbb{Z}_p$, we propose two different structures, which would be optimal for different kinds of computations.

The first type, a "p-adic ring with fixed modulus" would be just a wrapper for the ring $\mathbb{Z}/p^n\mathbb{Z}$ with additional functionality (such as Valuation()). This uses a dense representaiton, but it would be able to take advantage of fast integer arithmetic. Note that we completely disallow division, even exact division, since this is not well-defined. There would be some loss when printing these elements in the more typical p-adic way as a "power series" in p.

For the second type, we consider a "p-adic ring without variable modulus". Such rings would have two kinds of elements.

The first kind of element would be represented sparsely and with fixed precision: such an element would be represented by (1) its valuation as a nonnegative integer (the power of p), (2) the "unit part", and (3) the precision. We differentiate between the notion of *precision* (always relative to the valuation) and *modulus*, so that $3^5 + O(3^20)$ has modulus 20 and precision 15. Here, we may allow "exact division", adjusting the precision as needed. One has a choice to represent the unit part by either an integer (modulo p^n where n is the precision), a power series in p, or as a power series in $q = p^r$ where p^r is close to the size of a word. One does not really care if the unit is a unit, (until the user asks) in the first representation, so you only must normalize when you multiply, as $(p + O(p^2))(2p + O(p^2)) = 2p^2 + O(p^3)$. We could not decide which of these would be optimal, since they seem to have both advantages and disadvantages, so more research is required.

We propose a second kind of element which we call *lazy* elements (in analogy with so-called lazy power series rings). Here we would represent elements by a function, which would accept as input a nonnegative integer n and return a p-adic integer with precision p^n such that the inverse limit is well-defined. This could be a function, a user program, or any way that the user decides, and once computed the "lazy element" would remember its p-adic expansion. A special subclass of these elements would be reserved for $\mathbb{Q} \cap \mathbb{Z}_p$, which would instead replace the function by storing the accompanying rational number. In the future, perhaps a second instance would come from $\overline{\mathbb{Q}} \cap \mathbb{Z}_p$. When performing binary operations on two such elements, one would create a new lazy element which would remember the preceding two and the arithmetic operation, and similarly with unary operations; for elements of $\mathbb{Q} \cap \mathbb{Z}_p$, one would instead perform the ring operations there. The main advantage of this would analogous to using interval arithmetic for real numbers which yields "provably" correct results.

A choice of "default precision" for a p-adic ring seems to be worth avoiding, since for one, it is not canonical. Instead, one should have a "default printing precision", and then one can represent things like -1 as $2 + 2 \cdot 3 + 2 \cdot 3^2 + \cdots + O(2^{40})$; one might set this default printing precision to ∞ if you want to print all such information; then for elements of $\mathbb{Z}_p \cap \mathbb{Q}$, one should instead print the corresponding element of $\mathbb{Q}$.

There is always the problem of testing if an element is zero in a p-adic ring. For the first type with fixed modulus, this is perfectly well-defined. For the second type without variable modulus, for usual elements, one should say the element is zero if and only if it is zero up to the precision given, so for example $O(1)$ is equal to zero, and $1 + O(p^2)$ is equal to $1 + O(p)$. This is just like what one does for real numbers, by declaring they are equal if their difference is weakly zero, and then accepting the fact that there will be numerical instability which results. For lazy elements, one can compare the rational numbers and return an exception if they are just user-defined functions, and instead create a function IsWeaklyZero(n) which returns true if and only if it is zero to the precision p^n.

Our experience comes from working with Pari and Magma. What is implemented in KASH?

2. p-ADIC FIELDS

We have no new ideas here: we represent p-adic field elements as p-adic ring elements but allow negative valuation. The p-adic field and its elements should be different types than the p-adic ring and its elements.

The "usual" problems which come from loss of precision are usually due to unstable algorithms, such as using generic linear algebra functions rather than specialized ones. In the documentation, it would be a good idea to show and example of this and to issue a warning to the user that the p-adic field, as it is represented inexactly, really is not a field!

3. Extensions of p-adic rings and fields

Extensions which are defined by polynomials with $\mathbb{Q}_p$ coefficients are unstable: it is not necessarily possible to test a polynomial for irreducibility, and insufficient precision may not be enough to determine the extension up to isomorphism. One possibility (Magma) is to disallow it, another is to replace it with an (irreducible) polynomial over $\mathbb{Q}$ which we choose optimally and whose coefficients agree to the precision given. Perhaps then this extension can be represented internally as a tower of unramified and ramified extensions, then map back for output.

After much discussion, we decided that p-adic extension should be represented by the data of the completion of a number field (which we choose) and a prime ideal.

When printing elements, one should print power series in the uniformizer with coefficients in the unramified extensions.

[In Magma, when making very ramified extensions the default precision does not grow, which quickly causes the extension to degenerate; therefore a "default precision" is used, it needs to be multiplied by the ramification index.]

[We did not think about orders; though they make sense for global fields, for p-adic rings it is always cheap enough to work in the full ring of integers.]

3.1. **Things one might want to do with Local field extensions.**

(1) Galois closure
(2) Galois Groups
(3) Ramification Filtration of Galois group and filtration of fields
(4) Lattice of subfields
(5) Subfield generated by a set of elements (there's some argument about whether we can define such when in fact we're specifying not an element, but in fact a residue class)
(6) Compositum of two subfields of a given field
(7) Hom sets (local -¿ local, global -¿ local)
(8) Class Field theory of some sort

(9) Analytic functions (log, exp, powers, Artin-Hasse exponential, p-adic Γ)
(10) Unified notion of precision in power series over local fields
(11) Galois cohomology
(12) Stable linear algebra
(13) Factoring and extracting roots
(14) basic operations (add, subtract, multiply, divide)
(15) Database of local fields (Craig)

4. Other local fields

It should generalize!

5. Power series

We believe that our approach should also apply equally well to power series in one variable over an exact field. We also talked about representing the ring $\mathbb{Z}_p[[x]]$, which naturally comes about in several applications such as Monsky-Washnitzer cohomology; we noted that rather than representing these as pure power series over $\mathbb{Z}_p$ in one of its instantiations, it would be better for applications if one instead only allowed monomials that fit within a "linear bound": the term $p^a x^b$ is zero if and only if an integral linear functional on a, b is nonnegative. And its generalizations to power series in more than one variable. (The alternative, which stores lots of zeros, may not be so much more expensive if one is smart about knowing which terms are zero.)