Mathematics and Algorithms for Computer Algebra

July 9, 2003

7: Introduction to modular and p-adic methods

Modular and p-adic methods are often presented in texts on computer algebra only in the context of polynomial gcd computation and factorization. However, they are much more general than that, and my intention in this set of notes is briefly to introduce these methods in a general context. They will be discussed again later in the more specific context of gcds and factorization.

1 Computation by homomorphic images

The motivation for computing in a suitably chosen homomorphic image of the original problem domain is that the computation there is either easier or is possible, whereas in the original domain it was either harder or impossible. For example, any homomorphic image of the infinite ring of integers is a finite ring, and computation in a finite set is usually easier than in an infinite set; as we have already seen, a finite algorithm to find the solution of an equation over a finite set is, at least in principle, to simply test each element of the set, whereas over an infinite set this procedure does not give a finite algorithm. Moreover, there are homomorphic images of the integers that are fields although the integers themselves do not form a field, and finite fields are easy computational domains because division by any nonzero element is possible, but there is no need for frequent gcd computations as there is in a quotient field (field of fractions). Computing in a homomorphic image means computing in a quotient ring, i.e. modulo some ideal in the original ring. Frequently, the original problem ring is the ring of integers $\mathbb{Z}$, and hence the homomorphic image is $\mathbb{Z}_m$ for some $m \in \mathbb{Z}$, which might well be chosen to be prime.

Suppose the original problem is specified in an Ω-algebra A, and we can find a suitable homomorphism $\phi : A \to A'$ of Ω-algebras. Then rather than compute the value $h = \omega(a_1, \ldots, a_n)$ of some arity-n operator in A we compute its homomorphic image in A' indirectly via the defining property of a morphism:

$$r = \phi(\omega(a_1, \ldots, a_n)) = \omega(\phi(a_1), \ldots, \phi(a_n)).$$

Unfortunately, this does not give us the value of h, but only an equivalence class $[h]$ containing h specified by the kernel relation $\phi(h) = r$.

Recovering h from its equivalence class requires additional information, and there are two main practical methods to do it. The essence is always to ensure that the canonical representative of $[h]$ is h itself, which always requires information specific to the particular problem. However, to do this directly requires a homomorphism ϕ that does not project out too much of the structure of A, and in practice this usually makes the homomorphic image A' quite large and complicated and so loses most of the advantage of the homomorphic image method. One practical method, called the multiple homomorphic image (MHI) method, is to compute several homomorphic images of the desired solution and then take the intersection of the resulting equivalence classes. This essentially "lifts" the solution from several simple images to one much more complicated image. The tool used to do this in practice, at least in a Euclidean domain, is a Chinese Remainder Algorithm. The other practical method, called p-adic or Hensel lifting, is to compute a solution modulo one prime p, and then use this solution together with the original problem specification to lift to a solution modulo a power of p.

Before considering lifting algorithms it is appropriate to consider in more detail precisely why it is so much simpler to compute in a homomorphic image.

2 Homomorphic images of expressions

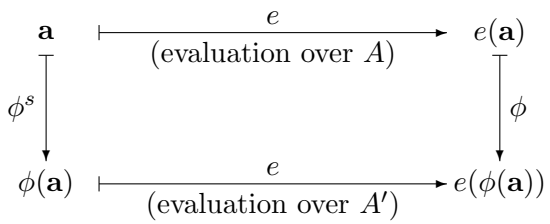
Let A be an Ω-algebra. Then a (finite) computation in A is the evaluation of an *expression* that combines elements of A using finitely many operations from Ω. The syntax and semantics of Ω-expressions can be defined formally – see Lipson, page 236. The important and useful point is that not only do morphisms preserve the values of operations, they also preserve the values of expressions, i.e. finite compositions of operations.

The following (vectorial) notation is convenient, in which $\mathbf{a}$ denotes "$a_1, \ldots, a_s$"; $\mathbf{a}'$ denotes "$a'_1, \ldots, a'_s$"; $\phi(\mathbf{a})$ denotes "$\phi(a_1), \ldots, \phi(a_s)$"; etc.

Theorem 1 (Fundamental Morphism Theorem) *Let $\phi : A \to A'$ be a morphism of Ω-algebras, and let $e(x_1, \ldots, x_s) = e(\mathbf{x})$ be an Ω-expression. Then*

$$\phi(e(\mathbf{a})) = e(\phi(\mathbf{a})).$$

In terms of a mapping diagram, this is



Proof is by induction on the number k of operations in the expression.

Basis ($k = 0$). Then $e(\mathbf{x}) = \mathbf{x}$, so that

$$\phi(e(\mathbf{a})) = \phi(\mathbf{a}) = e(\phi(\mathbf{a})).$$

Induction. For any $k \geq 0$, assume that the theorem is true for all Ω-expressions having $\leq k$ operation symbols, and let $e(\mathbf{x})$ have $k+1$ operation symbols. Then $e(\mathbf{x})$ has the form

$$e(\mathbf{x}) = \omega(e_1(\mathbf{x}), \ldots, e_n(\mathbf{x}))$$

for some n-ary operation symbol $\omega \in \Omega$, where $e_1(\mathbf{x}), \ldots, e_n(\mathbf{x})$ each have $\leq k$ operation symbols.

If $n = 0$ then $\omega()$ is a nullary operation, and $e(\mathbf{x}) = \omega()$. Then trivially $\phi(e(\mathbf{a})) = \phi(\omega()) = \omega() = \omega(\phi(\mathbf{a})) = e(\phi(\mathbf{a}))$ because ϕ preserves all operations, including nullary ones.

If $n > 0$ then

$$\begin{array}{rcll} \phi(e(\mathbf{a})) & = & \phi(\omega(e_1(\mathbf{a}), \ldots, e_n(\mathbf{a}))) & \text{definition of } e \\ & = & \omega(\phi(e_1(\mathbf{a})), \ldots, \phi(e_n(\mathbf{a}))) & \phi \text{ is a morphism} \\ & = & \omega(e_1(\phi(\mathbf{a})), \ldots, e_n(\phi(\mathbf{a}))) & \text{induction hypothesis} \\ & = & e(\phi(\mathbf{a})) & \text{definition of } e. \end{array}$$

□

Hence, when evaluating an expression mod m it is sufficient to reduce all initial values mod m and to compute mod m at all stages of the evaluation.

2.1 An example: "Casting out nines"

This is a "classical" method of checking decimal arithmetic: an integer in decimal representation can be reduced modulo 9 by simply summing its digits, and repeatedly summing the digits of the sum, until a single digit number results, discarding (casting out) any digits equal to 9 while doing so. The use of this is that if the numbers in an arithmetic equation involving only integers in decimal representation are each replaced by the sum of their digits then the equation must remain true (mod 9), and this can clearly be iterated until an equation involving single digit numbers results. This technique will quickly show up some, but not all, calculation errors. The justification lies in the Fundamental Morphism Theorem, as follows.

Let $a = (a_n, a_{n-1}, \ldots, a_0)_{10}$ be a decimal integer and let

$$a(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0,$$

so that $a = a(10)$ is an expression. Then by the Fundamental Morphism Theorem, $\phi(a(10)) = a(\phi(10))$. Let ϕ be the remainder mod 9 epimorphism $\mathbb{Z} \to \mathbb{Z}_9$. Then $r_9(a(10)) \equiv a(r_9(10)) \equiv a(1) \pmod 9$, and hence

$$(a_n, a_{n-1}, \ldots, a_0)_{10} \equiv a_n + a_{n-1} + \cdots + a_0 \pmod 9.$$

If the digits are repeatedly summed then the remainder mod 9 must finally result.

As an example, let us check the equation

$$43 \times 768 + 9571 = 42595.$$

Reducing each number mod 9 by casting out nines and summing digits twice gives

$$43 \times 768 + 571 \equiv 4255 \pmod 9 \;\Rightarrow$$

$$7 \times 21 + 13 \equiv 16 \pmod 9 \;\Rightarrow\; 7 \times 3 + 4 \equiv 7 \pmod 9.$$

Evaluating and again summing digits gives

$$25 \equiv 7 \pmod 9 \;\Rightarrow\; 7 \equiv 7 \pmod 9,$$

which is correct, and hence the original equation is true mod 9.

2.2 Division rings: partial algebras

Let R be a commutative ring. The divisibility relation defined on an integral domain applies also to commutative rings with zerodivisors: $b \mid a$ over R if $a = bc$ for some $c \in R$. A *division ring* is a ring in which division is defined for some elements: for b neither zero nor a zerodivisor, a/b is defined over R if $b \mid a$. Thus division in a ring is a *partial operation*, and so a division ring is a partial algebra. For example, $\mathbb{Z}$ is a division ring in which $6/3$ is defined and has value 2, but $7/3$ is not defined; in $\mathbb{Z}_m$, $a/b = ab^{-1}$ is defined if $\gcd(b, m) = 1$, otherwise b is a zerodivisor which cannot therefore be used as a divisor; in a field, $a/b = ab^{-1}$ is defined if $b \neq 0$. The reason for excluding zerodivisors as divisors is in order to have a unique value for a/b. For example, in $\mathbb{Z}_{12}$, which is a ring with zerodivisors and hence not a UFD, $6 = 3 \times 2 = 3 \times 6$. Therefore $3 \mid 6$, but is $6/3 = 2$ or 6?

The notions introduced previously of morphisms and expressions can be extended to include partial algebras provided that all operations are defined. However, it is usually possible to choose homomorphic images that avoid partial algebras, and division rings in particular, for example by ensuring that they are finite fields, so I will not elaborate this topics further – see Lipson, pp. 239–242 for further details.

In the rest of this set of notes I will consider only cases where the computational domain is primarily the integers, and so consider only computation in homomorphic images of the integers, which from our consideration of ideal theory we know must be $\mathbb{Z}_m$, the integers modulo m, for some m.

3 Integer computations by homomorphic images

An integer congruence $u \equiv r \pmod{m}$ has a unique solution in any range $\alpha \leq u < \alpha + m$, for any fixed but arbitrary real number α. There are two particularly important choices of α: (a) $\alpha = 0$, which gives the least positive solution u and (b) $\alpha = \lceil m/2 \rceil$, which gives the least absolute value solution.

Then the basic scheme for computing an integer value h via its homomorphic image r is the following:

1. Given a bound B such that either (a) $0 \leq h < B$ or (b) $|h| < B$, choose a modulus m such that (a) $m \geq B$ or (b) $m \geq 2B$ and all expressions are defined if working in a division ring, which is most easily achieved by choosing m to be prime, so that in fact the computation is performed over a field.

2. Reduce all input values mod m and then compute r mod m.

3. Solve the congruence $h \equiv r \pmod{m}$ for either (a) the least positive solution, or (b) the least absolute value solution.

3.1 Exact solution of linear equations

As an example, consider the problem of finding the exact solution in $\mathbb{Q}$ of a system $A\mathbf{x} = \mathbf{b}$ of linear equations over $\mathbb{Q}$, which can trivially be converted to equations over $\mathbb{Z}$. One method is to use Bareiss elimination directly, but here I want to consider a modular method. The solution can be conveniently separated into integer problems by using Cramer's Rule, which is that if A_j denotes the matrix A with its j^{th} column replaced by $\mathbf{b}$ then $x_j = |A_j|/|A|$, where all the determinants over $\mathbb{Z}$ must lie in $\mathbb{Z}$, and hence are appropriately evaluated by a homomorphic image scheme.

A bound B on the magnitudes of the determinants can be found using Hadamard's inequality (which also underlies the bound on the magnitude of a resultant – see the paper on "Some Useful Bounds" by M. Mignotte in Buchberger, Collins & Loos). Because this gives a magnitude bound, we choose a modulus $m \geq 2B$. The determinants are computed over $\mathbb{Z}_m$, which is possible for any m because a determinant is defined in terms only of ring operations, which do not include division. This can always be done by Bareiss elimination, or if m is chosen to be a prime then Gaussian elimination (over the finite field) could be used.

However, computing several determinants is not the most efficient way to solve this problem, and it is better to reduce the augmented matrix $(A, \mathbf{b})$ to echelon (i.e. essentially upper triangular) form. From this form, $|A|$ can be computed immediately, and the x_j can be computed by back-substitution. But these values for x_j are solutions of the problem over $\mathbb{Z}_m$, and we want solutions over $\mathbb{Q}$. However, the $|A_j|$ can now be computed indirectly as $|A_j| = |A|x_j$. The resulting values of $|A|$ and $|A_j|$ are then lifted to values in $\mathbb{Z}$, from which the solution over $\mathbb{Q}$ is computed as $x_j = |A_j|/|A|$. [If $\mathbf{b} = \mathbf{0}$ then a slightly different approach is required, because a solution exists only if $|A| = 0$ – for example, see Lipson, Appendix 2 to Chapter VIII.]

3.2 Computation of binomial coefficients

As another example, consider the problem of computing the binomial coefficient

$$
\begin{aligned}
{}^nC_r &= \frac{n!}{r!(n-r)!} = \frac{n(n-1)\cdots(n-r+1)}{r!} && (1)\\
&= \frac{n\cdots(n-(r-1)+1)(n-r+1)}{r(r-1)!} = {}^nC_{r-1}\frac{(n-r+1)}{r}. && (2)
\end{aligned}
$$

By definition, for $n \geq r \geq 0$ a binomial coefficient is an integer, even though the formula above involves division – therefore the division must be exact and the computation is possible over the ring of integers. A direct evaluation of the numerator and denominator in (1) can produce very large numbers, which are undesirable, but the recursive formula in (2) shows how to minimize the size of the intermediate numbers by performing multiplications and divisions together whilst keeping the divisions exact and the intermediate results integral. However, it is still possible for an intermediate product to have a value larger than the final result, and so potentially cause an unnecessary overflow.

An alternative is to compute over a finite field, in which case it is probably more efficient to compute the numerator and denominator separately and perform only one division. Suppose that we have an upper bound B on the value of the binomial coefficient, perhaps by using Stirling's approximation to the factorials. Then we choose a modulus $m \geq B$ that is prime. As a trivial example, let us compute

$$
{}^7C_3 = \frac{7.6.5}{3.2.1} < 50,
$$

supposing that we do not wish to use more than 2 decimal digits internally. Then a direct computation of the numerator (210) would overflow. We can choose any prime modulus $50 \leq m < 99$, so let us choose 53. Then in $\mathbb{Z}_{53}$ the numerator is

$$
7.6.5 = 210 \equiv 51 \pmod{53}
$$

and the denominator is $3.2.1 = 6$. To invert this, the appropriate Bézout identity is $1 = (-1).53 + 9.6$ and hence $6^{-1} \equiv 9 \pmod{53}$. Then

$$
{}^7C_3 = 51 \times 9 = 459 \equiv 35 \pmod{53}.
$$

Finally (in principle) we solve the congruence $h \equiv 35 \pmod{53}$ for the smallest positive solution 35, which is indeed the correct value of ${}^7C_3$, and has required only 2-digit intermediate values.

4 The Chinese Remainder Algorithm for the integers

In non-trivial computations of integer values they are frequently large, and to compute modulo an m that is even larger loses much of the advantage of modular computation – that it involves only small integers. The approach in this case, as outlined earlier, is to compute a family of homomorphic images each with distinct small moduli, and then take the intersection of the resulting equivalence classes of the true solution. The algorithm used in practice to do this can in essence be traced back to Chinese mathematicians around 2000 years ago, and hence is usually referred to as the Chinese Remainder Algorithm, because its general setting involves remaindering in a Euclidean domain.

The integer Chinese Remainder Problem (CRP) is essentially to determine the solutions to a system of n congruences of the form

$$u \equiv r_k \pmod{m_k} \quad (k = 0, 1, \ldots, n-1).$$

It is simplest to begin with the 2-congruence problem

$$u \equiv r \pmod{m}, \quad u \equiv s \pmod{n}$$

and to assume that m, n are relatively prime. [This assumption is not necessary, but it is true in practice and provides some simplification. See Appendix A.5 of Davenport *et al.* for the general version.] The solution to the first congruence is

$$u = r + \sigma m$$

for some integer σ. The problem is to restrict σ so that the second congruence is also satisfied, i.e. so that

$$r + \sigma m \equiv s \pmod{n} \;\Rightarrow\; \sigma m \equiv s - r \pmod{n}.$$

Now $m^{-1} \bmod n$ exists because (by assumption) $\gcd(m, n) = 1$, and therefore the solution is to choose $\sigma = (s-r)m^{-1} \pmod{n}$. This gives a solution $\bmod mn$:

Proposition 2

$$(u \equiv U \pmod{m} \text{ and } u \equiv U \pmod{n}) \iff u \equiv U \pmod{mn}.$$

Proof $\Leftarrow$ is trivial. To prove $\Rightarrow$, observe that the congruences imply that $u = U + ms = U + nt$ for some $s, t \in \mathbb{Z}$, and hence $ms = nt$. Therefore $n \,|\, ms \;\Rightarrow\; n \,|\, s$ because m, n are, by assumption, relatively prime, and so $s = rn$ for some $r \in \mathbb{Z}$. Therefore, $mrn = nt \;\Rightarrow\; t = rm$ since $\mathbb{Z}$ is an integral domain, and hence $u = U + mnr$. □

Now let us return to the case of more than two congruences

$$u \equiv r_k \pmod{m_k} \quad (k = 0, 1, \ldots, n-1).$$

We assume that the m_i are all relatively prime. Each r_i is usually in the range of its respective $\bmod m_i$ function (and if not it can be reduced so that it is) and hence the r_i are often referred to as "residues". The procedure is to replace the first pair of congruences by the equivalent congruence mod $m_0 m_1$, solve this congruence with the next one, and repeat this process iteratively until all the congruences have been solved. This is possible for the following reason.

If we define

$$M_k = \begin{cases} 1 & \text{if } k = 0, \\ \prod_{i=0}^{k-1} m_i & \text{otherwise,} \end{cases}$$

then it satisfies the following

Lemma 3 *M_k and m_k are relatively prime for $0 \le k < n$.*

Proof If $k = 0$ then $\gcd(M_k, m_k) = \gcd(1, m_k) = 1$. For $k > 0$ suppose to the contrary that $\gcd(M_k, m_k) = g$ is not a unit, and let p be a prime factor of g. Then $p \,|\, M_k$ and $p \,|\, m_k$. But $p \,|\, \prod_{i=0}^{k-1} m_i$ implies that $p \,|\, m_i$ for some $0 \le i \le k-1$, in which case p is a common divisor of m_i and m_k. But this contradicts the assumption that m_i and m_k are relatively prime; hence M_k and m_k must be relatively prime. □

Therefore, the pair of congruences being solved at any stage of the iteration always have relatively prime moduli.

The n-congruence Chinese Remainder Algorithm (CRA) is therefore this:

input: $u \equiv r_k \pmod{m_k}$ $(k = 0, 1, \ldots, n-1)$
$M := 1$;
$U := r_0 \bmod m_0$;
for $k := 1$ to $n - 1$ do
begin

$M := M \times m_{k-1}$;
$M_1 := M^{-1} \pmod{m_k}$;
$\sigma := [(r_k - (U \bmod m_k))M_1] \bmod m_k$;
$U := U + \sigma M$
$\{M = M_k; U \equiv r_i \pmod{m_i} \text{ for } i = 0, \ldots, k\}$
end.
output: $U \equiv u \pmod{M_n}$

This algorithm involves reductions $\bmod m_k$ that are not strictly necessary, but they reduce the complexity of the computation and so are desirable in practice.

After the k^{th} iteration this algorithm has computed $M = M_k$ and

$$U = U_k = \sigma_0 M_0 + \sigma_1 M_1 + \cdots + \sigma_k M_k = \sigma_0 + \sigma_1 m_0 + \cdots + \sigma_k m_0 m_1 \cdots m_{k-1}$$

where each σ_i is in the range of the $\bmod m_i$ function.

The following example is from Lipson, p. 258. If the above CRA is applied to compute the solution U of the following system of congruences over $\mathbb{Z}$,

$$u \equiv 1 \pmod{3}, \quad u \equiv 3 \pmod{5}, \quad u \equiv 0 \pmod{7}, \quad u \equiv 10 \pmod{11},$$

then the intermediate values are these:

k	r_k	m_k	M	M_1	σ	U
0	1	3	1			1
1	3	5	3	2	4	13
2	0	7	15	1	1	28
3	10	11	105	2	8	868

In principle, the above discussion applies to any Euclidean domain, but in practice a polynomial CRA normally uses an evaluation-interpolation technique.

5 Computation by Multiple Homomorphic Images (MHI)

Most of the previous discussion on integer computations by a single homomorphic image generalizes to a scheme using multiple homomorphic images and Chinese remaindering, if the previous single modulus is replaced by the

product of the multiple moduli. One of the goals of using an MHI scheme is to avoid the need for long integer arithmetic, but the larger the moduli the fewer need to be used, so they are usually chosen to be as large as possible but smaller than the word size of the computer used, i.e. no larger than the base used for long integers. Moreover, they are usually chosen to be prime, so that the homomorphic images are finite fields.

5.1 Computing appropriate primes

The problem is to compute the N (perhaps 10 or 20) *largest* primes $\leq W$, where W is the word size of the computer. This can be achieved by using the Sieve of Eratosthenes, which is usually the most efficient method for computing a set of primes together. First an auxiliary list of primes $\leq \sqrt{W}$ is computed. Then the multiples of these primes are deleted from the list $\{W, W-1, \ldots, W-\Delta W\}$, where ΔW is chosen large enough to contain at least N primes. Then the integers remaining must be prime, because if an integer x is composite then it must have a factor, and hence a prime factor, $\leq \sqrt{x}$. But is this scheme feasible?

Useful information is provided[1] by the celebrated

Theorem 4 (Prime Number Theorem) *The number $n(x)$ of primes $\leq$ x is asymptotically $x/\log x$.*

Hence $dn/dx \approx 1/\log x$ and the number of primes in an interval ΔW around W is approximately $\Delta W/\log W$. Therefore, we should take ΔW to be "safely" in excess of $N \log W$ to ensure that it contains at least N primes. Also, the number of primes $\leq \sqrt{W}$ should not be too much larger than $\sqrt{W}/\log\sqrt{W}$. For example, with $W = 2^{31}-1$ and $N = 20$ we need $\Delta W \geq N \log W = 20 \log 2^{31} = 429$, so 1000 might be a good choice, and the size of the table of small primes needs to be a bit larger than $\sqrt{2^{31}}/\log\sqrt{2^{31}} \approx 4310$. These estimates give perfectly feasible sizes, and the computation of the 20 primes need be performed once only and only the primes themselves retained for subsequent use. Moreover, for fixed n the running time of the algorithm clearly increases as $\sqrt{W}$ rather than W. In fact, Knuth (1981, p. 390) gives a table of the 10 primes less than a wide range of word sizes.

[1] Lipson, p. 282; Hardy and Wright, *The Theory of Numbers*, 4^{th} ed. Oxford, 1960, Section 1.8; for finite x the theorem slightly underestimates the number of primes.

6 p-adic methods

Modular methods work by solving a problem modulo several small primes and then lifting to a solution modulo the product of those primes by using the Chinese Remainer Algorithm. This lifting is quite independent of the original problem. The p-adic approach is to solve a problem, which must be expressible in terms of a system of algebraic equations, modulo one small prime p, and then lift to a solution modulo successive powers of p. This requires additional use of the original system of equations at the lifting stage (whereas Chinese Remaindering does not). The p-adic approach was introduced by K. Hensel around 1900, and p-adic lifting is usually called "Hensel lifting". Hensel's original lifting algorithm lifted from p^k to p^{k+1} at each step, and is therefore called *linear lifting* because it is linear in the power of p. In 1969, H. Zassenhaus introduced a lifting that lifts from p^k to p^{2k} at each step and is therefore called *quadratic lifting*.

6.1 The p-adic integers

We have seen how any integer n can be represented with respect to any chosen base integer B. Suppose that the base is the prime number p; then the representation has the form

$$n = \sum_{i=0}^{k-1} n_i p^i.$$

To make this representation unique we can require that $n_{k-1} \neq 0$, and then the integer n has k digits in base p. Now recall that one can compute the lowest order (i.e. rightmost) digit of n as

$$n_0 = n \bmod p.$$

Therefore, knowledge of the value of $n \bmod p$ is equivalent to knowledge of the first digit in the positional representation of n in base p. Similarly,

$$n_0 + n_1 p = n \bmod p^2,$$

and hence knowledge of the value of $n \bmod p^2$ is equivalent to knowledge of the first two digits, and so on. Therefore, linear Hensel lifting of the value of n from p^k to p^{k+1} is equivalent to determining one more higher order digit, the $(k+1)^{th}$, in the positional representation of n in base p. This is the sense in which the term "p-adic" appears normally to be used in the

context of computer algebra. However, Hensel developed a theory of "p-adic numbers", which is a little more subtle than simply a finite representation in a prime base. This subtlety is not required in order simply to compute integers by Hensel lifting, but it is required for other applications, and is interesting in its own right anyway.

Let p be a prime integer, and define on $\mathbb{Z}$ the "p-adic norm"

$$\begin{aligned} |a|_p &= p^{-r} \text{ if } a \neq 0,\ p^r \mid a \text{ and } p^{r+1} \nmid a, \\ |0|_p &= 0. \end{aligned}$$

Thus, the p-adic norm $|a|_p$ of $a \in \mathbb{Z}$ is smaller the larger the power of p that a contains as a factor. As a simple illustration, the 2-adic and 3-adic norms of the integers 0–19 are shown in the following table:

integer	**2-adic norm**
$0, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19$	$2^0 = 1$
$2, 6, 10, 14, 18$	$2^{-1} = 1/2$
$4, 12$	$2^{-2} = 1/4$
8	$2^{-3} = 1/8$
16	$2^{-4} = 1/16$
integer	**3-adic norm**
$0, 1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19$	$3^0 = 1$
$3, 6, 12, 15$	$3^{-1} = 1/3$
$9, 18$	$3^{-2} = 1/9$

With respect to the p-adic norm, an infinite series of the form

$$n = \sum_{i=0}^{\infty} n_i p^i, \quad n_i \in \mathbb{Z},$$

always converges. The set of such infinite series is called the "p-adic integers", which I will denote by $\mathcal{Z}_p$. It constitutes the completion of $\mathbb{Z}$ with respect to the p-adic norm. I will denote $a \in \mathcal{Z}_p$ as $(a)_p$ when it is necessary to distinguish between a and its p-adic representation. Exactly as for the base-p representation of an element of $\mathbb{Z}$, each element $a \in \mathcal{Z}_p$ can be written uniquely in the canonical form

$$a = \sum_{i=0}^{\infty} a_i p^i, \quad a_i \in \{0, 1, \ldots, p-1\}\ (= \mathbb{Z}_p),$$

and, if

$$b = \sum_{i=0}^{\infty} b_i p^i,$$

addition and multiplication are defined by

$$a + b = \sum_{i=0}^{\infty} (a_i + b_i) p^i, \quad ab = \sum_{k=0}^{\infty} \left(\sum_{i+j=k} a_i b_j \right) p^k.$$

In general, these formulae do not give a canonical representation because of "carries" between digit positions (powers of p), but the results are easily canonicalized.

The p-adic representation of any finite non-negative integer will have only a finite number of non-zero digits, after which all the remaining higher-order digits to the left will be zero. One of the differences between the p-adic representation and the standard representation in base p appears when we consider negative integers. The non-canonical p-adic representation

$$(0)_p = p + \sum_{i=1}^{\infty} (p-1) p^i$$

is canonicalized to 0 because the carry from the first digit position propagates throughout the infinite sequence of digits, and it is also obvious that the sum is formally equal to 0. Hence the representation of $-a$ is obtained by subtracting a from $(0)_p$ (and then canonicalizing if necessary). In particular,

$$-1 = \sum_{i=0}^{\infty} (p-1) p^i,$$

which has the maximal digit in each of the infinitely many digit positions. More generally, if

$$(a)_p = \sum_{i=k}^{\infty} a_i p^i$$

then the *canonical* representation of $-a$ is given by

$$(-a)_p = 0_p p^k - a = (p - a_k) p^k + \sum_{i=k+1}^{\infty} (p - 1 - a_i) p^i,$$

and if $(a)_p$ has a finite number of non-zero digits (e.g. a is a non-negative integer) then $(-a)_p$ has an infinite number of leading maximal digits $(p-1)$

to the left. In fact, $(-a)_p$ is precisely the p's complement representation of $-a$ without the truncation that is normally necessary. Subtraction is defined in the standard group-theoretic way in terms of negation as

$$a - b = a + (-b).$$

6.2 The p-adic numbers

The ring $\mathcal{Z}_p$ is integral and therefore has a quotient field, called the *field of p-adic numbers* and denoted $\mathcal{Q}_p$. Each non-zero $\alpha \in \mathcal{Q}_p$ can be written *uniquely* in the form

$$\alpha = \sum_{i=r}^{\infty} \alpha_i p^i, \quad r \in \mathbb{Z}, \quad \alpha_r \neq 0.$$

If α is a non-integral rational number then $r < 0$. This representation makes manifest the fact that $\mathcal{Z}_p \subset \mathcal{Q}_p$.

This representation is equivalent to the standard positional representation of non-integral numbers, used most commonly in the form of decimal fractions, except that in general the standard representation has an infinite sequence of non-zero digits *after* the point,[2] i.e. $r = -\infty$. By contrast, a p-adic representation in general requires an infinite sequence of non-zero digits *before* the point.

The formulae for addition, multiplication, negation and subtraction in $\mathcal{Q}_p$ are essentially the same as those for $\mathcal{Z}_p$. The formula for division in $\mathcal{Q}_p$ is in principle the same as the formula for division of formal power series (as in fact are those for addition and multiplication), but it does not generally produce a canonical representation, and a careful modular interpretation of the formula is required. Hensel lifting, to be considered in the next section, is probably the best approach in those cases where it works. Another strange property of p-adic numbers is that *some*, but certainly not all, algebraic equations are solved by p-adic numbers, and some algebraic irrationals are p-adic *integers*, of which some can best be computed by Hensel lifting.

I will finish this section with some examples of 2-adic numbers, which generalize two's complement representation, and are taken from exercise 31

[2]By "point" I mean the point that separates digits multiplying a negative power of the base from those multiplying a non-negative power, which is called the "decimal point" in conventional decimal representation. In countries where the native language is not (a dialect of) English (such as Brazil!) this point is frequently represented as a comma.

to §4.1 of Knuth, 1981:

$$\begin{array}{rclrcl} 7 &=& (\ldots 000000000000111)_2 & -7 &=& (\ldots 111111111111001)_2, \\ \frac{1}{7} &=& (\ldots 110110110110111)_2 & -\frac{1}{7} &=& (\ldots 001001001001001)_2, \\ \frac{7}{4} &=& (\ldots 000000000000001.11)_2 & \frac{1}{10} &=& (\ldots 110011001100110.1)_2, \\ \multicolumn{6}{c}{\sqrt{-7} = (\ldots 100000010110101)_2 \text{ or } (\ldots 011111101001011)_2.} \end{array}$$

Notice that the positional representation extends infinitely to the left but only finitely to the right of the point. Addition, subtraction and multiplication follow the normal rules but extended indefinitely to the left. The numbers $\frac{1}{7}$, $-\frac{1}{7}$ and $\sqrt{-7}$ are all 2-adic integers, because they are not divisible by any negative power of 2, whereas $\frac{7}{4}$ and $\frac{1}{10}$ are not 2-adic integers because they are divisible respectively by 2^{-2} and 2^{-1}, and so have respectively 2 and 1 digits to the right of the point.

6.3 Newton's iteration and Hensel lifting

In his paper "Computing by Homomorphic Images" in Buchberger, Collins & Loos, M. Lauer presents Hensel lifting in the general context of the solution of a system of algebraic equations over a commutative ring modulo a finitely generated ideal. My presentation is based on Lauer's, but restricted to one equation in one unknown, so as to focus on the principles involved, and in particular the relationship to the Newton-Raphson iteration for improving an approximation to a root of a univariate equation (which extends quite easily to much more general situations).

In fact, the essence of the Newton-Raphson iteration is the following

Lemma 5 (Taylor Expansion) *Let R be a commutative ring, $f \in R[x]$ and let y be another indeterminate; then*

$$f(x+y) = f(x) + y\frac{df}{dx} + F$$

where $F \in R[x][y]$ and $F \equiv 0 \pmod{(y)^2}$.

Here, $(y)^2$ denotes the square of the ideal (y) in $R[x][y]$; in other words, F is a polynomial in x, y having the form $y^2G(x,y)$.

Theorem 6 (Hensel) *Let R be a commutative ring, (p) a principal ideal generated by a prime $p \in R$, $f \in R[x]$ and let $a \in R$ satisfy*

$$f(a) \equiv 0 \pmod{p}.$$

Let $f'(a) = \frac{df}{dx}(a) \in R$ *be invertible mod* p. *Then for each positive integer* k *there exists* $a^{(k)} \in R$ *such that*

$$f(a^{(k)}) \equiv 0 \pmod{p^k} \quad \textit{and} \quad a^{(k)} \equiv a \pmod{p}.$$

Proof is by induction on k. Clearly, the proposition is true for $k = 1$ by taking $a^{(1)} = a$, so let $k \geq 1$ and assume that the proposition is true for k. Then

$$f(a^{(k)}) \equiv 0 \pmod{p^k} \;\Rightarrow\; f(a^{(k)}) = mp^k$$

for some $m \in R$. Set $a^{(k+1)} = a^{(k)} + bp^k$, $b \in R$, so that clearly $a^{(k+1)} \equiv a \pmod{p}$ if $a^{(k)} \equiv a \pmod{p}$. Then the task is to choose b so that $f(a^{(k+1)}) \equiv 0 \pmod{p^{k+1}}$. By Taylor expansion

$$f(a^{(k+1)}) = f(a^{(k)} + bp^k) = f(a^{(k)}) + bp^k f'(a^{(k)}) \pmod{p^{k+1}}.$$

(In fact, the above equation is true mod p^{2k}, which is the basis of quadratic lifting!) Hence

$$f(a^{(k+1)}) = mp^k + bp^k f'(a^{(k)}) = (m + bf'(a^{(k)}))p^k \pmod{p^{k+1}}.$$

The right side vanishes mod p^{k+1} if

$$m + bf'(a^{(k)}) \equiv 0 \pmod{p}$$

i.e.

$$b = -m(f'(a^{(1)}))^{-1} \pmod{p},$$

where by assumption the inverse exists. □

An inverse modulo a prime can always be computed by using Bézout's identity via the extended Euclidean algorithm. The above proof can be made into an algorithm to explicitly lift a solution as follows.

input: $a^{(1)}$ such that $f(a^{(1)}) \equiv 0 \pmod{p}$
$f_1 := (f'(a^{(1)}))^{-1} \pmod{p}$;
for $k := 1$ to $k_{\max} - 1$ do
  $a^{(k+1)} = r_{p^{(k+1)}}(a^{(k)} - f(a^{(k)})f_1)$.
output: $a^{(k_{\max})}$ such that $f(a^{(k_{\max})}) \equiv 0 \pmod{p^{k_{\max}}}$

This is obviously the Newton-Raphson algorithm, except that the difference between successive approximations is *increasing* by multiples of increasing

powers of p. Whilst the resulting sequence would not converge using a conventional norm, it does converge in the p-adic norm. The lifting is continued until sufficient digits of the p-adic representation of a have been computed, e.g. until p^k is larger than an upper bound on a.

As an example, I will use the above Hensel lifting algorithm to compute the 2-adic representation of $1/7$ by solving the equation $f(x) = 7x - 1 = 0$. Then $f'(x) = 7$, and the appropriate Bézout identity is $1 = 1.7 + (-3).2$, and hence $f_1 = 1/7 = 1 \pmod 2$, from which also follows that $7.1 - 1 \equiv 0 \pmod 2$, and hence $a^{(1)} = 1$. The iteration formula is

$$a^{(k+1)} = r_{p^{(k+1)}}(a^{(k)} - (7a^{(k)} - 1) \times 1) = r_{p^{(k+1)}}(1 - 6a^{(k)})$$

and digit $(k+1)$ of $a^{(k+1)}$ is given by $(a^{(k+1)} - a^{(k)})/p^k$. The lifting algorithm gives the following results:

k	$a^{(k)}$	$1 - 6a^{(k)}$	$a^{(k+1)}$	digit $(k+1)$
0			1	1
1	1	-5	3	1
2	3	-17	7	1
3	7	-41	7	0
4	7	-41	23	1
5	23	-137	55	1
6	55	-329	55	0

Note that the Hensel lifting algorithm cannot be applied if $f'(a^{(1)}) \equiv 0 \pmod p$, because then $f'(a^{(1)})$ is not invertible. Moreover, this condition implies that *all* derivatives vanish $\bmod p$ and that no derivative is invertible modulo *any power* of p. Bearing this in mind, let us consider trying to compute the 2-adic representation of $\sqrt{-7}$ by solving any equation of the form $f(x) = g(x)(x^2 + 7) = 0$ and ignoring any solution of $g(x) = 0$. Then the initial solution must satisfy $(a^{(1)})^2 + 1 \equiv 0 \pmod 2 \Rightarrow a^{(1)} = 1$ uniquely. But $f'(x) = g(x).2x + g'(x)(x^2 + 7) \equiv g'(x)(x^2 + 1) \pmod 2$, and hence $f'(a^{(1)}) \equiv 0 \pmod 2$ for any choice of $g(x)$.

This means that Hensel lifting cannot be used to compute the 2-adic representations of $\frac{7}{4}$, $\frac{1}{10}$ or $\sqrt{-7}$ quoted above, and some more complicated method is required, such as undetermined coefficients – see the exercises for an example.

7 Exercises

No questions in this final set of exercises for Part 1 of the course will be assessed, in order to provide a clean break between Parts 1 and 2.

1. By "casting out nines" show which of the possible values shown for the following expression over $\mathbb{Z}$ are wrong:

$$57 \times 394 + 89354 = 111812,\ 118112,\ 111820.$$

Explain why the method does not prove that the value of an expression is correct.

2. Compute by hand some less trivial binomial coefficients, such as ${}^{20}C_7$ and ${}^{23}C_{17}$, using modular methods. [Check your computations by computer if you wish.]

3. [Lipson] Solve over $\mathbb{Q}$ the following two systems of equations:

$$(a)\quad \begin{aligned} 2x - 7y &= 4, \\ 5x - 3y &= -1; \end{aligned} \qquad (b)\quad \begin{aligned} x - 2y + 3z &= -2, \\ 2x + y - 5z &= 4, \\ 4x - 3y + z &= 0. \end{aligned}$$

Use first the single homomorphic image method for one suitable prime modulus, and then the multiple homomorphic images method for a suitable set of prime moduli ≤ 13.

4. Construct the 2-adic representations for 4 and 10. Hence verify, by explicit p-adic addition and multiplication, the 2-adic representations given in the notes for -7, $\pm\frac{1}{7}$, $\frac{7}{4}$, $\frac{1}{10}$ and $\sqrt{-7}$.

5. Solve the equation $x^2 + 1 = 0$ in $\mathcal{Z}_5$, i.e. construct two representations of the imaginary number i as 5-adic integers, by Hensel lifting. Is this also possible in $\mathcal{Z}_3$, or even in $\mathcal{Q}_3$? Prove your assertion.

6. Compute the 2-adic representations, as given in the notes, for $\sqrt{-7}$ by the method of undetermined coefficients. This is not difficult, but tedious by hand, and is an ideal task for a computer algebra system. For example, here are the instructions necessary in REDUCE 3.4 to compute the first 7 digits. The term "`tail`" is to avoid trying to compute more digits than are available accurately from the assumed form of solution "`a`". This code could easily be converted into a procedure (algorithm) to compute an arbitrary number of digits – try to do so.

```
a := a0 + 2^1 a1 + 2^2 a2 + 2^3 a3 + 2^4 a4 + 2^5 a5
 + 2^6 a6 + 2^7 tail;
eqn := a^2 + 7;  on modular;
setmod 2;  eqn;  a0 := 1;   % Choose a0 to make eqn = 0
setmod 4;  eqn;             % Identically 0 - the problem
setmod 8;  eqn;  a1 := 0;   % Two choices possible for a1
setmod 16;  eqn;  a2 := 1;  % Choose a2 to make eqn = 0,
setmod 32;  eqn;  a3 := 0;  % etc.
setmod 64;  eqn;  a4 := 1;
setmod 128;  eqn;  a5 := 1;
setmod 256;  eqn;  a6 := 0;
setmod 512;  eqn;  off modular;  eqn;
% The remaining digits are clearly not all zero.
```