The Similarities (and Differences) between Polynomials and Integers

Susan Landau*and Neil Immerman†
Department of Computer Science
University of Massachusetts
Amherst, Mass. 01003

July 16, 1993

Abstract

The purpose of this paper is to examine the two domains of the integers and the polynomials, in an attempt to understand the nature of complexity in these very basic situations. Can we formalize the integer algorithms which shed light on the polynomial domain, and vice versa? When will the casting of one in the other speed up an existing algorithm? Why do some problems not lend themselves to this kind of speed-up?

We give several simple and natural theorems that show how problems in one domain can be embedded in the other, and we examine the complexity-theoretic consequences of these embeddings. We also prove several results on the impossibility of solving integer problems by mimicking their polynomial counterparts.

1 Introduction

It is a fact frequently remarked upon that polynomials and integers share a number of characteristics. Usually the Fast Fourier Transform is then

*Supported by NSF grants DMS-8807202 and CCR-9204630.
†Supported by NSF grant CCR-9207797.

given as an example of this phenomenon, and the problem of multiplying polynomials is shown to be embedable as a problem in multiplying integers.

In 1974, Borodin and Moenck [6], extended examples of Cabay [10], Brown [9] and Collins [13], and examined the issue of fast modular transforms (evaluating mod p for integer problems, evaluating at integer points for polynomial problems) as a general technique for algorithmic speed-up in the two domains. In 1982, shortly before the Lenstra, Lenstra, Lovász [17] polynomial time algorithm for polynomial factorization, Adleman and Odlyzko gave a reduction from polynomial irreducibility testing and factorization to integer primality testing and factorization respectively [2]. More recently, Char, Geddes and Gonnet [11], and independently, Schönhage [22], gave a fast algorithm for computing polynomial gcds which relies on computing integer gcds, and then interpolating to find the polynomial.

These ideas can only be carried so far, because of the problem with carries. Can we formalize the integer algorithms which shed light on the polynomial domain, and vice versa? When will the casting of one in the other be useful, that is, when will it speed up an existing algorithm? Why do some problems not lend themselves to this kind of speed-up?

We give several simple and natural theorems that show how problems in one domain can be embedded in the other, and we examine the complexity-theoretic consequences of these embeddings. Where integer algorithms are unexpectedly faster than their polynomial analogues, we examine how the structure of the problems benefits from the structure of the integers. Finally we prove several results on the impossibility of solving integer problems by mimicking their polynomial counterparts.

2 The Size of the Problem

The two domains of interest in this paper are Z, the ring of integers, and $Z[x]$, the ring of polynomials with integer coefficients. If $a \in Z$ is an n-digit integer, a can be written as $\pm a_{n-1}10^{n-1} + \ldots + a_0$, with the $a_i \in \{0, 1, \ldots 9\}$. Thus the size of a, written $[\![a]\!]$ is n. Similarly, if $a(x) \in Z[x]$ is a polynomial, then $a(x) = a_{n-1}x^{n-1} + \ldots + a_0$, with the a_i in Z. Clearly there are two parts to the size of a polynomial: its degree and its coefficients. For the purpose of comparing the complexity of the two domains, we will assume that the polynomials have bounded coefficient size, with $|a_i| \leq 2^c$ for some constant

c. Thus the size of $a(x)$, written $[\![a(x)]\!]$, is nc. With these bounds in mind, we have the following table of well-known results:

problem	integer $a =$ $a_{n-1}10^{n-1} + \ldots + a_0$	polynomial $a(x) =$ $a_{n-1}x^{n-1} + \ldots + a_0$, $\lvert a_i \rvert < 2^c$
multiplication	$n \log n \log\log n$ [23][3]	$cn \log n(\log n + \log(c/n \log n))$ [3]
division	$n \log n \log\log n$ [24]	$cn \log n(\log n + \log(c/n \log n))$ [3]
gcd	$n \log^2 n$ [21]	$cn \log n(\log n + \log(c/n \log n))$ [3]
irreducibility	$n^{\log\log n}$ [1], [12]	polynomial in n, c [26]
factorization	$2^{n^{1/4+\epsilon}}$ [20], [25]	$n^{9+\epsilon} + n^{7+\epsilon}c$[17]

Figure 1: Deterministic Time Complexity of Some Basic Problems

There are a number of things to note in this table. For the first three problems (which happen to be the basic building blocks), the algorithms for the integer version of the problems is $\log n$ times faster than the algorithms for the polynomial versions. For the last two – these are problems which in some sense seek to undo the effect of carries – the polynomial problems have exponentially faster algorithms.

3 Embedding Integers as Polynomials, and Vice Versa

In this section we attempt to formalize the idea that polynomial and integer algorithms are very similar implementations of the same domain. A natural question to ask is whether an algorithm in one domain translates to an algorithm in the other. We begin with the question of embedding integer problems as polynomial ones.

A naive answer to the question of "Can one model any integer problem as a polynomial problem?" would be to embed the integer as the constant term of the polynomial. From a complexity viewpoint, this does not seem to be a useful approach. Instead we view an n-digit integer as a polynomial of degree $n-1$, in the variable "10". Of course, we need not limit ourselves to base 10; instead we will let the base c be part of the input. Then $a = a_{n-1}c^{n-1} + \ldots + a_0$ can be viewed as a polynomial $a_{n-1}x^{n-1} + \ldots + a_0$.

Let $\Gamma(z_1, \ldots, z_r)$ be a straight-line program each line of which is of the form

$$v_i := t_1 \text{ op } t_2$$

where op$\in \{+, -, \times\}$ and t_i is a previously defined variable v_s, or one of the variables z_l.

Suppose for the remainder of this section that

$$\Gamma \text{ consists of } k \text{ lines, } j \text{ of which are multiplications.} \tag{3.1}$$

We will think of the variable assigned in the last line of Γ as its output. Thus, for any ring R, Γ induces a map

$$\Gamma_R : R^r \to R.$$

In the next two theorems we will consider two invertible mappings between Z and $Z[x]$.

Theorem 3.2 *Let Γ be a straight-line program as above. Let $c \geq 2$ be a natural number, and define $\varphi_c : Z \to Z[x]$ to be the map that takes the integer $a_{n-1}c^{n-1} + \ldots + a_0$ in c-ary notation to the polynomial $a_{n-1}x^{n-1} + \ldots + a_0$. Thus φ_c^{-1} exists, and is given by $\varphi_c^{-1}(p(x)) = p(c)$. Then the following diagram commutes:*

$$\begin{array}{ccc} Z^r & \xrightarrow{(\varphi_c)^r} & (Z[x])^r \\ \Gamma_Z \downarrow & & \downarrow \Gamma_{Z[x]} \\ Z & \xleftarrow{\varphi_c^{-1}} & Z[x] \end{array}$$

Proof This follows immediately from the fact that $(f(x) \text{ op } g(x))|_a = f(a) \text{ op } g(a)$ where op $\in \{+, -, \times\}$. ∎

Corollary 3.3 *For any $c \geq 2$, the complexity of computing Γ_Z on r $n-$digit integers written in $c-$ary is no more than the complexity of computing $\Gamma_{Z[x]}$ on r polynomials of degree n with coefficients of absolute value less than c, plus the time to evaluate a polynomial of degree no more than $2^j n$ with coefficients of absolute value less than $2^j(c + k \log n)$ at c.*

Proof Clearly the translation of the r $n-$bit integers takes $O(rn \log c)$ steps. If $\Gamma_{Z[x]}$ is a k-step straight-line program with j multiplications, each of the $\log c-$bit coefficients may have grown in size. The worst case scenario occurs when the $k-j$ non-multiplication operations occur first, followed by the j multiplications. In the first addition, the coefficient can have at most $c + \log n$ digits. If we repeat this another $k-j-1$ times, the coefficients will have grown to at most $c + (k-j)\log n$ bits. The polynomials involved will still be of degree n at most. Now the first multiplication will yield $2(c + (k-j)\log n) + \log n$ bits per coefficient, and $2n$ terms. The next multiplication will yield $(2(2(c + (k-j)\log n) + \log n)) + \log 2n$ bits per coefficient, and $4n$ terms. By the j^{th} multiplication, we will have:

$$2(\ldots(2(2(c + (k-j)\log n) + \log n) + \log 2n) + \ldots) + \log 2^{j-2}n$$

bits per coefficient, with $2^j n$ terms. But

$$\begin{aligned} 2(\ldots(2(2(c + (k-j)\log n) + \log n) + \log 2n) + \ldots) + \log 2^{j-2}n &= \\ 2^j(c + (k-j)\log n) + 2^{j-1}\log n + 2^{j-2}\log 2n + \ldots + \log 2^{j-1}n &< \\ 2^j(c + (k-j)\log n) + 2^j \log n &< \\ 2^j(c + k\log n). & \end{aligned}$$

to a length of $2^j(c + k\log n)$ bits at most. There will be $2^j n$ such coefficients, and thus we have the bound of the theorem. ∎

We can also embed any polynomial problem as an integer one, but we have to work a little harder to do that. We need to consider how to represent negative as well as positive coefficients of the polynomial. Fortunately there is some work by Avizienis [4] on signed digit representations of integers which handles this problem.

Define the map

$$\rho_m(a_0, \ldots, a_{n-1}) = \Sigma_0^{n-1} a_i m^i,$$

for $|a_i| \leq m$. We call the notation of the right balanced $m-$ary.

To go from ordinary $m-$ary notation to balanced $m-$ary is simply a digit-by-digit copy; clearly in $\mathcal{NC}^0$. To go the other way is a single subtraction of $m-$ary integers, thus it is in $\mathcal{AC}^0$. The advantage of balanced $m-$ary notation is that it gives very efficient implementations of $+, -, \times$:

Theorem 3.4 *(Borodin, Cook, Pippenger) [5] Under balanced $m-$ary notation, addition and negation of integers can be implemented in NC^0, and multiplication in NC^1.*

Balanced $m-$ary notation allows us to use integers to represent polynomials in a way that easily represents both positive and negative coefficients. Let Z_m denote the set of formal sums, $\Sigma_{i=0}^{n-1} a_i m^i$. The following is a map from polynomials to balanced $m-$ary representation of the integers:

$$\theta_m : \Sigma_{i=0}^{n-1} a_i x^i \rightarrow \Sigma_{i=0}^{n-1} a_i m^i.$$

It is not hard to see that this map is invertible on the set of polynomials with coefficients bounded in absolute value by by $\frac{m-1}{2}$. This provides the key to a map between polynomials and integers that gives an analogue to Theorem 3.2.

Theorem 3.5 *Let Γ be a straight-line program as above. Let $c \geq 2$ be a natural number and let $Z_c[x]$ be the set of polynomials with integer coefficients bounded by c in absolute value. Define $\rho : Z[x] \rightarrow Z$ to be the map that takes the polynomial $a_{n-1}x^{n-1} + \ldots + a_0$, to the integer $\Sigma_{i=0}^{n-1} a_i m^i$, where $m = (cn2^{k+1-j})^{2^j}$. Then $(\rho)^{-1}$ exists, and is given by*

$$(\rho)^{-1}(\Sigma_{i=0}^{n-1} a_i m^i) = a_{n-1}x^{n-1} + \ldots + a_0.$$

The following diagram commutes:

$$\begin{array}{ccccc}
Z_c[x] & \xrightarrow{\theta_m} & Z_m & \xrightarrow{\rho_m} & Z \\
\Gamma_{Z[x]} \downarrow & & \downarrow \Gamma_{Z_m} & & \downarrow \Gamma_Z \\
Z_m[x] & \xleftarrow{(\theta_m)^{-1}} & Z_m & \xleftarrow{(\rho_m)^{-1}} & Z
\end{array}$$

Proof The idea is that we translate each polynomial to an integer, do the computation, and translate back. We will do this via a series of maps, each one of which is one-to-one. Let $\rho = \rho_m \circ \theta_m$, and let

$$\theta_m(a_{n-1}x^{n-1} + \ldots + a_0) = \Sigma_{i=0}^{n-1} a_i m^i$$

where $m = (cn2^{k+1-j})^{2^j}$. Furthermore, let

$$(\theta_m)^{-1}(\Sigma a_i m^i) = a_n x^n + \ldots + a_0.$$

Here $\rho_m : Z_m \to Z$ is the map defined by evaluating the formal sum. In order to show that $\Gamma_{Z[x]}$ and $\theta_m^{-1}(\rho_m^{-1}(\Gamma_Z(\rho_m(\theta_m))))$ commute, it suffices to show (1) that $(\theta_m)^{-1}\Gamma_{Z_m}\theta_m$ and $\Gamma_{Z[x]}$ commute, and (2) that $(\theta_m)^{-1}\Gamma_Z\theta_m$ and Γ_{Z_m} commute. That the second diagram commutes is well-known from the work of Avizienis [4].

To show that the first diagram commutes, it will suffice to show that we have picked m sufficiently large that there are no carries in the balanced $m-$ary arithmetic, and thus that the coefficients are acting independently. Thus the map is one to one.

Since Γ has k steps, of which at most j are multiplications, the worst case scenario occurs when the $k-j$ non-multiplication operations occur first, followed by the j multiplications. Since each addition or subtraction at most doubles the absolute value of a coefficient, we have a tight upper bound,

$$c_0 \;=\; c2^{k-j}$$

for the magnitude of the coefficients after these $k-j$ operations.

Thus the largest possible coefficients would arise from taking the polyomial,

$$f \;=\; c_0(x^n + x^{n-1} + \cdots + x + 1)$$

and squaring it j times. Each squaring of the polynomial increases the magnitude of coefficients by at most squaring them and then multiplying by the degree. Thus an over estimate of the size of the resulting coefficients is that

$$c_1 \;<\; (cn2^{k+1-j})^{2^j} = m$$

By the way we have chosen m, the magnitude of any resulting coefficients is less than $\frac{m}{2}$. Thus $\theta_m^{-1}(\Gamma_{Z_m}(\theta_m))$ commutes with $\Gamma_{Z[x]}$. ∎

Corollary 3.6 *For any $c \geq 2$, the complexity of computing $\Gamma_{Z[x]}$ on r degree n polynomials with coefficients bounded by c in absolute value is no more than the complexity of computing Γ_Z on r $n-$digit integers written in base m, where $m = (cn2^{k+1-j})^{2^j}$.*

Proof Obvious, from the above comments. ∎

One way to think of Theorem 3.5 is that it is a very simple technique which enables one to implement polynomial arithmetic on a machine with large integer arithmetic, bearing in mind that the computations are far from efficient. Thus if one wanted to compute the product of the polynomials

$$x^2 + x, x^3 + 1, 2x + 5,$$

one could do this by doing arithmetic base m, for a suitably chosen m. Using the algorithm above, we discover $k = 2, j = 2, n = 3, c = 5$, and thus $m = 30^4$. We find:

$$x^2 + 2x \to 2(30^4) + (30^4)^2$$
$$x^3 + 1 \to 1 + (30^4)^3$$
$$2x + 5 \to 5 + 2(30^4)$$

The product is:

$$10(30^4) + 9(30^4)^2 + 27(30^4)^3 + 10(30^4)^4 + 5(30^4)^5 + 2(30^4)^6,$$

which translates, of course, to:

$$10x + 9x^2 + 27x^3 + 10x^4 + 5x^5 + 2^6,$$

which is of course the product of the three polynomials.

To decrease the size of the blow-up, we could occasionally reduce by converting to an integer, and then back to a polynomial.

On the surface both Theorem 3.2 and Theorem 3.5 appear to have exponential blow-ups. In fact, the transformations behave radically differently with regard to this issue. In transforming an integer problem to a polynomial one, the exponential blow-up occurs when writing the bits of the result. In the transformation of the polynomial problem to an integer one, the exponential blow-up occurs in the size of the input to the polynomial problem. Thus the blow-up happens even if the result turns out not to have an exponential number of digits.

Looking at it another way, we have shown an exponential blow-up is sufficient to mimic any polynomial problem as an integer one. If we could show that mimicking any polynomial problem as an integer problem requires

such a blow-up, we would be showing that polynomial problems are easier than their integer counterparts. That would be interesting indeed.

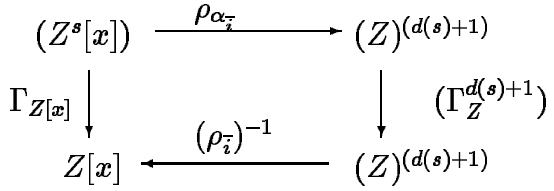
It is not true. In looking at the table on page 2, there are several examples in which the integer problem is faster than its polynomial counterpart. The explanation for this behavior is quite simple. In those problems the carries are performed during the operations, and not held until the end. The resulting data compression enables a speed-up of the algorithm. One can make use of the fast integer problem to develop a faster technique for the polynomial problem. For example, Char, Geddes and Gonnet [11], and Schönhage [22] computed gcds of degree n polynomials by computing $n+1$ different integer gcds and interpolating. They did this via a probabilistic algorithm.

Suppose you wish to compute the $\gcd(g(x), h(x))$, where $g(x), h(x)$ are polynomials with coefficients in Z. Then Char et al first compute "primitive" representatives for $g(x)$ and $h(x)$, polynomials $g_1(x)$ and $h_1(x)$ such that $g(x) = zg_1(x)$, $h(x) = zh_1(x)$, and the coefficients of $g(x)$ and $h(x)$ are relatively prime. Then they randomly pick values a_i for x, and compute $b_i = \gcd(g(a_i), h(a_i))$. If their sample space is reasonably sized, then with high probability the value they compute b_i will in fact equal the $\gcd(g(x), h(x))$ evaluated at (a_i). Using the values they have computed, they interpolate to find the gcd. With high probability, their solution is correct.

Note that a deterministic algorithm can be used for in the above, using large substituted values. However, the algorithm is faster in practice when values are chosen at random in a range of values smaller than what can be guaranteed to work deterministically, cf. Theorem 4.3.

Interpolation provides another general framework for mapping between polynomials and integers:

Theorem 3.7 *Let Γ be a straight-line program as above. Let $d(s)$ be a bound such that for all polynomials $f_1, \ldots, f_r \in Z[x]$ of degree at most s, $\Gamma_{Z[x]}[f_1, \ldots, f_r]$ has degree at most $d(s)$. Let $\bar{i} = i_0, \ldots, i_{d(s)}$ be any $d(s)+1$ distinct integers. Let $\alpha_{\bar{i}}(p(x)) = (p(i_0), \ldots, p(i_{d(s)})$, and let $\alpha_{\bar{i}}(Z[x]) \rightarrow Z[x]$ be the polynomial interpolation function. Let $Z^s[x]$ be the polynomials in $Z[x]$ of degree at most s. Then the following diagram commutes:*



Proof This follows immediately from the fact that any polynomial of degree k can be determined by computing its value on $k+1$ points and interpolating. ∎

Corollary 3.8 *The complexity of computing $\Gamma_{Z[x]}$ on r degree n polynomials with coefficients bounded by c in absolute value is no more than the complexity of computing $\Gamma_Z^{d(s)+1}$ on $d(s)+1$ r-tuples of n-digit integers written base c, plus the time to evaluate a polynomial of degree no more than $d(s)$ with coefficients of absolute value less than $2^j(c+k\log n)$ at c.*

Multiplication, division and gcd share the good fortune of having a single small degree polynomial as a result of the computation. Thus one way to perform the operations for polynomials is to use Theorem 3.7. Although the problem of factorization also has small degree polynomials for its results, the number of factors may be as large as n. The problem of recombination after substituting integer values – which factors to pair with which – admits exponentially many possibilities. Thus Theorem 3.7 does not seem to lead to a fast technique for polynomial factorization.

4 Adding Division

In this section we make some observations which imply that two out of three of the above transformations are still valid when the straight line programs include divisions.

The version of the division algorithm we use will be equally appropriate for polynomials and integers:

Division	Integers	Polynomials
Input:	$a, b \in Z, b \neq 0$	$a, b \in Z[x], b$ monic
Output:	$q, r \in Z, -\frac{\lvert b\rvert}{2} < r \leq \frac{\lvert b\rvert}{2}$ such that $a = bq + r$	$q, r \in Z[x]$, $\deg(r) < \deg(b)$ such that $a = bq + r$

We begin with the case that does not work. Recall the map defined in Theorem 3.2 taking integers base c to polynomials:

$$\begin{array}{rcl} \varphi_c : Z & \rightarrow & Z[x] \\ a_{n-1}c^{n-1} + \ldots + a_0 & \mapsto & a_{n-1}x^{n-1} + \ldots + a_0 \end{array}$$

Obviously this map does not preserve division as the following example with $c = 10$ shows:

$$\begin{array}{rcl} 4x^2 & = & (2x-9)(2x+9) + 81 \quad \text{but,} \\ 400 & = & 13 \cdot 29 + 23 \end{array}$$

Here is the point: the map φ_c is an isomorphism preserving $+$ and $\cdot$, but it does not preserve the ordering relation. For example, the polynomial $2x + 9$ is greater than the constant polynomial 81, because the ordering on polynomials is by degrees. That does not imply that $29 > 81$.

Thus, it seems nearly impossible to preserve integer division in this setting. On the other hand, there do exist efficient and essentially identical polynomial and integer reciprocal algorithms [3][§8.2,8.3]. These algorithms provide bounds that will be useful in adding division to the mappings from polynomials to integers.

Recall the mapping from $\rho : Z[x] \rightarrow Z$ defined in Theorem 3.5 by

$$a_{n-1}x^{n-1} + \ldots + a_0 \mapsto a_{n-1}m^{n-1} + \ldots + a_0$$

where m was defined to be an appropriately large value. Since we can take m to be quite large, the ordering is preserved when we need it and ρ preserves division:

Theorem 4.2 *Theorem 3.5 still holds for straightline programs including d divisions by monic polynomials in addition to the j multiplications and the $k - j$ additions or subtractions as long as the bound m is modified so that $m = (cn^2 2^{k+4d+1})^{4^{j+d}n}$.*

Proof We have to show that for sufficiently large m, if P and D are polynomials with D monic such that the result of the polynomial division algorithm is

$$P = Q \cdot D + R$$

procedure RECIPROCAL$\left(\sum_{i=0}^{k-1} a_i x^i\right)$

if $k = 1$ **then return**$(1/a_0)$

else { $q(x) \leftarrow$ RECIPROCAL$\left(\sum_{i=k/2}^{k-1} a_i x^{i-k/2}\right)$

$r(x) \leftarrow 2q(x)x^{(3/2)k-2} - (q(x))^2 \left(\sum_{i=0}^{k-1} a_i x^i\right)$

return$(\lfloor r(x)/x^{k-2} \rfloor)$

Figure 4.1: Algorithm From [3] to compute $\lfloor x^{2k-2} / \sum_{i=0}^{k-1} a_i x^i \rfloor$

then it follows that the result of the integer division algorithm for $\rho(P)/\rho(D)$ is

$$\rho(P) \;=\; \rho(Q) \cdot \rho(D) + \rho(R)$$

Since the values of the division algorithm are unique and ρ preserves additions and multiplications, it suffices to show that $|\rho(R)| < |\rho(D)|/2$.

The idea is that even after divisions the coefficients remain less than half of the new value of m. Since the degree of R is less than the degree of D it follows that $|\rho(R)| < |\rho(D)|/2$, as desired.

The proof is: use the algorithm from [3] (see Figure 4.1) to compute the reciprocal of D: $C = \lfloor x^n/D \rfloor$. The polynomial reciprocal is computed using $2\log(\text{degree}(P))$ additions and the same number of multiplications. Some truncations are also performed but these of course do not increase the size of any coefficients. The quotient, Q is then computed by multiplying C times P and doing one more truncation. Another subtraction gives R as well. Recall that the degree is bounded by $2^j n$. Thus the bound follows from Theorem 3.5 by substituting $k + 4d\log(2^j n)$ for k and $j + 2d\log(2^j n)$ for j. ∎

For interpolation, the following lemma implies that as long as the substituted values are large enough, Theorem 3.7 remains true when the straight line program includes divisions.

Theorem 4.3 *Theorem 3.7 goes through with straight-line programs including division as one of the allowable operations as long as the substituted values are sufficiently large, i.e.,* $|\alpha| \geq (cn^2 2^{k+4d+1})^{4^{j+d}n}$.

Lemma 4.4 *Let $a(x), b(x) \in Z[x]$ with $b(x)$ monic. Let $\alpha \in Z$ satisfy $|\alpha| \geq (cn^2 2^{k+4d+1})^{4^{j+d_n}}$. Suppose that Q and R are the quotient and remainder resulting from dividing $a(\alpha)$ by $b(\alpha)$. Then $Q = q(\alpha)$ and $R = r(\alpha)$, where $q(x)$ and $r(x)$ are the quotient and remainder resulting from dividing $a(x)$ by $b(x)$, as polynomials.*

Proof Since the division algorithm produces unique answers, we know that $Q, R, q(x), r(x)$ are uniquely determined by the following equations and inequalities:

$$\begin{aligned} a(\alpha) &= Qb(\alpha) + R \\ -|b(\alpha)|/2 &< R \leq |b|/2 \\ a(x) &= q(x)b(x) + r(x) \\ \deg(r) &< \deg(b) \end{aligned}$$

It thus suffices to show that

$$|r(\alpha)| < |b(\alpha)|/2.$$

Since $deg(r) < deg(b)$, this follows when α is sufficiently large. In particular, the value $m = (cn^2 2^{k+4d+1})^{4^{j+d_n}}$ from Theorem 4.2 suffices. ∎

5 Polynomials are Simpler than Integers

We know that carries complicate matters. But it is one thing to assert something that every first grade teacher can agree upon, and quite another to prove it.

The factorization of an integer and its analogous polynomial do not always match. Because multiplication in the integer domain involves carries, the factorization of an integer does not necessarily map to a factorization of the related polynomial. The flip situation occurs when a composite polynomial maps to a prime integer. This can happen if all but one of the factors of the polynomial evaluate to ± 1 when substituting the base b for the variable x.

On the other hand, sometimes integers and polynomials do behave analogously: $144 = 12 \times 12$ and $x^2 + 4x + 4 = (x + 2)(x + 2)$. However, 144

also has the factorization 9×16 but $x^2 + 4x + 4 \neq (x-1)(x+6)$. Base 9, we have $144_{10} = 170_9 = 10_9 \times 17_9$, and $x^2 + 7x = x(x+7)$. There is no base b such that all possible factorizations of 144 have analogous polynomial factorizations.

This is a result of unique factorization. The two distinct factorizations we have written for 144_{10} are factorizations into composite integers. The factorizations for the polynomial are factorizations into irreducible polynomials. Thus there are several factorizations for the integer, but only a single factorization for the polynomial.

If we try bases 2 or 3, we get some prime factors from the factorization of the analogous polynomial. In base 2, we have $144_{10} = 10010000_2$ which gives rise to $x^7 + x^4 = x^4(x^3 + 1)$, or the integer factorization $2^4 \times 9$. In base 3 we get $144_{10} = 12100_3$, which leads to $x^4 + 2x^3 + x^2$. That factors into $x^2(x+1)^2$, which leads to the integer factorization $3^2 4^2$. Larger bases than 3 do not yield prime factors.

The interweaving that results from carries is the underlying reason. Let C_b^n be the ratio of multiplications which have carries in an $n \times n$ multiplication table base b to all multiplications in an $n \times n$ table. Then for all bases $b \geq 2$, $\lim_{n \to \infty} C_b^n = 1$. More interesting to show would be that most products involve a carry in all reasonably sized bases, (bases $b \leq \log n$).

We could mimic an integer factorization problem as a polynomial one by splitting the coefficients of the polynomial in such a way as to undo the effect of the carries. For example, if one wants to factor 1729 by viewing it as a polynomial, one would consider the factorization of a number of different polynomials: $x^3+7x^2+2x+9, x^3+6x^3+12x+9, x^3+5x^2+22x+9$, etc. The difficulty with such an approach is that it leads to an exponential number of possibilities. This is because of:

Theorem 5.1 *[18] Let $f(x) = g_1(x) \ldots g_k(x)$, where $f, g_1, \ldots, g_k$ are polynomials with rational integer coefficients. If $f(x) = a_n x^n + \ldots + a_0$ and $g_i(x) = b_m x^m + \ldots + b_0$, then $|b_i| \leq \binom{m}{i}(\Sigma_i a_i^2)^{1/2}$.*

This bound is tight. Let $f(x)$ be the polynomial to be factored. If it has a factor $g(x)$ of reasonably large degree, say $n/2$, the bound on its coefficients b_i will be $|b_i| \leq 2^{n/2}$. Splitting the coefficients of $f(x)$ across the polynomial (by writing $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$ as $(a_n - 1)x^n + (a_{n-1} + 10)x^{n-1} +$

$\ldots + a_0, (a_n - 1)x^n + a_{n-1}x^{n-1} + (a_{n-2} + 100)x^{n-2} + \ldots + a_0$, etc) leads to an exponential number of possibilities to consider.

The rest of this section is devoted to making precise some of the above ideas, and proving some small results.

The polynomials we are interested in are of a specific type, those whose coefficients run between 0 and $b-1$ for some base b. Let P_b^n be the set of all polynomials base b whose coefficients lie between 0 and $b-1$. Let N_b^n be that subset of P_b^n whose leading and constant terms are both non-zero. (Since we are interested in situations in the limit, we let n be at least two.) Finally let $I_b^n(x)$ be the subset of N_b^n which consists of irreducible polynomials. There is the following long-standing but unproved conjecture:

Conjecture 5.2 *[19] For all bases $b \geq 2$, the* $\lim_{n\to\infty}[\![I_b^n(x)]\!]/[\![N_b^n(x)]\!] = 1$.

By contrast "most" integers factor. The Prime Number Theorem states:

Theorem 5.3 *Let $\pi(x)$ be the number of primes less than x. Then* $\lim_{x\to\infty} \frac{\pi(x) ln\, x}{x} = 1$.

A surprising fact is that sometimes integer irreducibility (primality) can carry over to polynomial irreducibility. Brillhart, Filaseta and Odlyzko [8] have shown:

Theorem 5.4 *If a prime p is expressed in the number system with base $b \geq 2$ as $p = \Sigma_{k=0}^n a_k b^k, 0 \leq a_k \leq b-1$, then the polynomial $\Sigma_{k=0}^n a_k x^k$ is irreducible.*

It is well known that there is no polynomial over Z which represents only primes. Thus the converse of Theorem 5.4 is false. But it is false in an even stronger manner. It is not just the case that an irreducible polynomial sometimes maps to a composite. Let Irreducible Polynomials Base b ($\mathcal{IPB}$) be the hypothesis that most polynomials base b are irreducible, as per Conjecture 5.2. Under that hypothesis, we find that an irreducible polynomial maps to a composite almost all the time.

Observation 5.5 *Assuming $\mathcal{IPB}$, most composites do not arise from reducible polynomials. More precisely, let $\rho_b^n : N_b^n \to Z$ be the usual map from polynomials to integers base b. Then for all bases $b \geq 2$,*

$$\lim_{n\to\infty} Prob(\rho_b^n(f(x)) \text{ is prime} | f \text{ is irreducible}) = 0.$$

We now turn our attention to a potentially simpler problem than factorization, that of computing "square" parts. Long before it was known how to factor polynomials in polynomial time, there was a fast algorithm for computing the "square" part of a polynomial; if $f(x) = g^2(x)h(x)$, with $(g(x), h(x)) = 1$, then $\gcd(f(x), f'(x)) = g(x)$. By iterating such a procedure, in polynomial time one could determine a factorization $f(x) = g_1(x)g_2^2(x)\ldots g_k^k(x)$ where $(g_i(x), g_j(x)) = 1$ if $i \neq j$. But no similarly fast algorithm has been found for integers. We say an integer n is "squarefree" if all the prime factors of n have exponent 1. The present fastest algorithm to compute a squarefree decomposition of integers gives a reduction to $\varphi(n)$ [16]. This presently takes exponential time to compute.

We have seen polynomial algorithms translate to integer algorithms, and vice versa. In its first blush, polynomial factorization appears to shed no light on integer factorization. So it also seems that the fast polynomial multiple factor decomposition also sheds no light on the integer situation. It has been known for over a century that a non-trivial portion of integers have a square factor:

Theorem 5.6 *(Gegenbauer)[15] Let $Q(x)$ be the number of squarefree numbers not exceeding x. Then $Q(x) = \frac{6x}{\pi^2} + O(\sqrt{x})$.*

By contrast, under $\mathcal{IPB}$, most polynomials do not. Consider the following diagram:

$$\begin{array}{ccc} Z_b^n & \xrightarrow{\rho_b^{n-1}} & N_b^n(x) \\ \Big\downarrow \mathrm{g} & & \Big\downarrow h = \gcd(f(x), f'(x)) \\ Z_b^n & \xleftarrow{\rho_b^n} & N_b^n(x) \end{array}$$

where Z_b^n is the set of $n+1$-digit integers base b with first and last digit non-zero.

Theorem 5.7 *Assuming $\mathcal{IPB}$, for all bases $b \geq 2$, if the above diagram commutes, then $g(x) = 1$ almost everywhere.*

Proof Since Z_b^n is the set of $n+1$-base b integers with non-zero leading and last bits, $\rho_b^{n-1}(Z_b^n) = N_b^n$. Now by $\mathcal{IPB}$, we know that $\lim_{n\to\infty} [\![I_b^n]\!]/[\![N_b^n]\!] =$

1. In a strong sense, most polynomials will be squarefree, and thus for all $b \geq 2$, $h(\rho_b^n)^{-1}))(Z_b^n) = 1$ almost everywhere. Then $g = \rho_b^n(h((\rho_b^n)^{-1}))(Z_b^n) = 1$ almost everywhere. ∎

On the other hand, by Theorem 5.6, a non-trivial proportion of the integers will have square factors. Thus,

Corollary 5.8 *There is no algorithm $g : Z \to Z$ that "simulates" the computation for polynomials and computes square factors, even if we computed this map for polynomially many different bases b.*

What can we say without hypotheses? It is possible to construct an infinite set of examples such that the simulation of integer arithmetic by polynomials does not lead to computing square factors:

Theorem 5.9 *For each base $b \geq 2$, there is an infinite set $\mathcal{M}_b$ of integers $m_{b,k}$ such that the integer version of differentiation and computing gcd's does not yield the computation of the largest square factor dividing $m_{b,k}$.*

Proof We do this by constructing for each base b and each integer k, an integer $m_{b,k}$ such that each $m_{b,k} = s_{b,k}^2$ is $2k-1$ digits long, and $s_{b,k}$ does not divide the $\gcd(m_{b,k}, (m_{b,k})')$ where n' is the simulation in integers of the process of differentiation in polynomials.

Let $m_{b,k} = (b^k - 1)^2 = b^{2k} - 2b^k + 1$. In base b notation, that would be written as:

digit :	$b-1$	$\ldots$	$b-1$	$b-2$	0	$\ldots$	0	1
position :	$2k-1$	$\ldots$	$k+2$	$k+1$	k	$\ldots$	2	1

Thus

$$\begin{aligned} m'_{b,k} &= (2k-1)(b-1)b^{2k-2} + \ldots + (b-1)(k+2)b^k + (b-2)b^k \\ &= (b-1)b^k\text{junk} + (b-2)b^k \\ &= ((b-1)\text{junk} + (b-2))b^k. \end{aligned}$$

Now $m_{b,k} = (b-1)(b^{k-1} + \ldots + b + 1)$. Thus it is easy to verify that $\gcd(m_{b,k}, m'_{b,k}) = 1$.

∎

This theorem is of limited value. While it shows that for every base b and every bit length $2k-1$, there will be an integer in that base of that length for which the integer simulation of the polynomial gcd trick for computing squarefree decomposition does not compute a squarefree decomposition for the integer, it does not rule out the possibility that transforming to a different base might do the trick. Furthermore, this set of integers is sparse.

We say a set of integers $\mathcal{M}_b$ is dense, if for all but finitely many values of k, $|\{m \in \mathcal{M}_b | |m| < b^k\}| \geq c^k$ for some constant $c > 1$. More useful than Theorem 5.9 would be a version that shows there is dense set of integers for which the integer version of the polynomial algorithm for computing the square decomposition fails.

6 Open Questions

This paper is an introduction into what we hope will become a useful technique for proving upper and lower bounds for integer and polynomial problems. Our theorems are sparse, our open questions are almost everywhere dense. We leave the reader with several of what appear to be the easier of these, since solving the more difficult ones would be tantamount to proving a superpolynomial lower bound for the problem of integer factorization, or showing that $\mathcal{P} \neq \mathcal{NP}$.

1. (a) Improve the bound in Theorem 3.2, or show that in some sufficiently general framework, this is best possible.

   (b) Improve the bound in Theorem 3.5, or show that in some sufficiently general framework, this is best possible.

2. It would be interesting to compute the bounds necessary for Lemma 4.4 and compare them with the probabilistic bounds of [11] and [22] for polynomial gcds.

3. The set of questions we are asking here can be naturally extended to the questions of parallel complexity. Here one of the most striking examples is the $\mathcal{NC}$ algorithms for polynomial gcds using subresultants. No $\mathcal{NC}$ integer algorithm is known. In this context it is natural to examine the Chinese Remainder Theorem as a transformation for parallelizing integer computations.

4. Let $\mathcal{C}(n)$ be the set of all composite integers less than n. Show that

   $$\text{Carry}(n) = \{m \in \mathcal{C}(n) | m = \Pi p_i^{a_i} \text{ has a carry for all bases } c, 2 \leq c \leq \log m\}$$

   has more than $p(\log n)$ elements for all polynomials $p(x)$.

5. Without assuming $\mathcal{IPB}$, prove that Observation 5.5 holds for a dense set of composites.

6. (a) Show that for a dense set of integers $M_k = \{m | |m| < 2^k\}$, for every base b, $2 \leq b \leq \sqrt{m}$ and for all but finitely many values of k, the integer version of the polynomial algorithm for computing square decompositions fails.

   (b) Prove that computing the square decomposition of an integer is computationally equivalent to factoring.

References

[1] L. Adleman, C. Pomerance, R. Rumely, "On Distinguishing Prime Numbers from Composites," *Math. Ann.*, Vol. 117 (1983), pp. 173-206.

[2] L. Adleman and A. Odlyzko, "Irreducibility Testing and Factorization of Polynomials," *Math. Comp.*, Vol. 41 (1983), pp. 699-709.

[3] A. Aho, J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[4] A. Avizienis, "Signed-Digit Number Representations for Fast Parallel Arithmetic," *IRE Transactions on Electronic Computers*, Vol. 10 (1961), pp. 389-400.

[5] A. Borodin, S. Cook, and N. Pippenger, "Parallel Computation for Well-Endowed Rings and Space-Bounded Probabilistic Machines," *Information and Control,* **58**, 1983, (113-136).

[6] A. Borodin and R. Moenck, "Fast Modular Transforms," *J. Comput. Sys. Sci.*, Vol. 8 (1973), pp. 366-386.

[7] A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier (1975).

[8] J. Brillhart, M. Filaseta and A. Odlyzko, "On an Irreducibility Theorem of A. Cohn," *Can. J. Math*, Vol. 33 (1981), pp. 1055-1059.

[9] W. Brown, "On Euclid's Algorithm and the computation of polynomial greatest common divisors," *J. Assoc. Comput. Mach.*, Vol. 18, pp. 478-504.

[10] S. Cabay, "Exact Solutions of Linear Equations," *Proc. of the Second Sympositum on Symbolic and Algebraic Manipulation*, 1971.

[11] B. Char, K. Geddes, and G. Gonnet, "GCDHEU: Heuristic Polynomial GCD Algorithm Based on Integer GCD Computation," *J. Symb. Comput.*, (1989), pp. 31-45.

[12] H. Cohen, A. Lenstra, "Implementation of a New Primality Test," *Math. Comp.*, Vol. 48 (1987), pp. 103-121.

[13] G. Collins, "The Calculation of Multivariate Polynomial Resultants," *J. Assoc. Comput. Mach.*, Vol. 18, pp. 515-532.

[14] P. Gallagher, "The Large Sieve and Probabilistic Galois Theory," *Proc. Symp. in Pure Math.* (1972), pp. 92-101.

[15] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, 1971.

[16] S. Landau, "Some Remarks on Computing the Square Parts of Integers," *Information and Computation* , Vol. 78, No. 3 (1988), pp. 246-253.

[17] A.K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, "Factoring Polynomials with Rational Coefficients," *Mathematische Annelan,* 261 (1982), pp. 513-534.

[MST] Yishay Mansour, Baruch Schieber, Prason Tiwari, "Lower Bounds for Computations with the Floor Operation," *SIAM J. Comput.*20(2), 315-327.

[MST] Yishay Mansour, Baruch Schieber, Prason Tiwari, "A Lower Bound for Integer Greatest Common Divisor Computations," *J. Assoc. Comput. Mach.*38(2) (1991), 453-471.

[18] M. Mignotte, "An Inequality about Factors of Polynomials", *Math. Comp.*, Vol. 28 (1974), pp. 1153-1157.

[19] A. Odlyzko, private communication.

[Pan] Victor Pan, "Complexity of Computations with Matrices and Polynomials," *SIAM Review* 34(2) (1992), 225-262.

[20] J. Pollard, "Theorems on Factorization and Primality Testing," *Proc. Cambridge Philos. Soc.*, Vol. 76 (1974), pp. 521-528.

[21] A. Schönhage, "Schnelle Berechnung von Kettenbruchentwiklungen," *Acta Informatica*, Vol. 1 (1971), pp. 139-144.

[22] A. Schönhage, "Probabilistic Computation of Integer Polynomial GCDs," *Journal of Algorithms*, Vol. 9 (1988), pp. 365-371.

[23] A. Schönhage and V. Strassen, "Schnelle Multiplikation grosser Zahlen," *Computing*, Vol. 7 (1971), pp. 281-292.

[24] M. Sieveking, "An Algorithm for Division of Power Series," *Computing*, Vol. 10 (1972), pp. 153-156.

[25] V. Strassen, "Einege Resultate über Berechnungskeomplexität", *Jahresber. Deutch. Math.-Verein*, Vol. 78 (1976-77), pp. 1-8.

[26] P. Weinberger, "Finding the Number of Factors of a Polynomial," *J. Algorithms*, vol. 5 (1984), pp. 180-186.