Journal of Symbolic Computation **26** (1998), 339–341.

COMPOSING POWER SERIES OVER A FINITE RING IN ESSENTIALLY LINEAR TIME

DANIEL J. BERNSTEIN

Abstract. Fix a finite commutative ring R. Let u and v be power series over R, with $v(0) = 0$. This paper presents an algorithm that computes the first n terms of the composition $u(v)$, given the first n terms of u and v, in $n^{1+o(1)}$ ring operations. The algorithm is very fast in practice when R has small characteristic.

1. Introduction

Let f be a polynomial over a commutative ring R, and let g be an element of $R[x]/x^n$. The point of this paper is a simple new algorithm to compute $f(g)$ when R has finite characteristic.

For prime characteristic, see section 2. For prime-power characteristic more generally, see section 3. For other characteristics, use the Chinese Remainder Theorem, as in [5, equation 4.3.2–9].

Applications. The problem of computing $f(g)$, under the restrictions $\deg f < n$ and $g(0) = 0$, is known as **order-n power series composition**. Here's the point: given power series u and v over R, with $v(0) = 0$, define $f = u(z) \bmod z^n$ and $g = v(x) \bmod x^n$; then $f(g) = u(v(x)) \bmod x^n$.

Power series composition is the bottleneck in **reversion** and **iteration** of power series. See [2] and [5, section 4.7].

Previous work. Brent and Kung describe two power series composition algorithms in [2]. The first algorithm computes $f(g)$ in n^α ring operations for some $\alpha > 1.5$, depending on the speed of matrix multiplication. This algorithm can be applied to the more general problem of **modular composition**; see [4]. The second algorithm computes $f(g)$ in about $n^{1.5}$ ring operations, provided that g' is invertible and that all primes up to about $\sqrt{n}$ are cancellable in R.

My algorithm takes $n^{1+o(1)}$ ring operations if R is fixed. It is the method of choice for power series composition over rings whose characteristic is a product of small primes; in particular, fields of small prime characteristic. The second Brent-Kung algorithm remains the fastest method for fields of large prime characteristic.

2. Prime characteristic

Fix a ring R of prime characteristic p. I will reduce the problem of computing $f(g)$, with $\deg f < d$ and $g \in R[x]/x^n$, to a sequence of p subproblems with d and n both reduced by a factor of p. Write $m = \lceil n/p \rceil$.

Date: 19971007.

1991 *Mathematics Subject Classification.* Primary 13P99.

The author was supported by the National Science Foundation under grant DMS–9600083.

Notice that g^p is a polynomial in x^p: there is a polynomial $h \in R[y]/y^m$ with $g^p = h(x^p)$. The coefficient of y^j in h is the pth power of the coefficient of x^j in g.

Split f as $f(z) = f_0(z^p) + zf_1(z^p) + \cdots + z^{p-1}f_{p-1}(z^p)$. Compute each $f_j(h)$ recursively by the same procedure; substitute $y \mapsto x^p$ into $f_j(h)$ to obtain $f_j(g^p)$; finally apply Horner's rule to evaluate $f(g) = f_0(g^p) + \cdots + g^{p-1}f_{p-1}(g^p)$.

The recursion stops when d is sufficiently small. For example, $f(g)$ is simply $f(0)$ when $d = 1$.

3. Prime-power characteristic

Fix a ring R of characteristic p^k, with p prime and $k \geq 1$.

Write $A = R[x]/x^n$. Also write $m = \lceil n/p \rceil$ and $B = R[y]/y^m$. Embed B into A by $y \mapsto x^p$; this embedding, which amounts to some copying inside the computer, is not stated explicitly in the following algorithm.

Algorithm C. Given $f \in R[z]$ and $g \in A$, to compute $f(g + \epsilon)$ in the ring $A[\epsilon]/(\epsilon^k, p\epsilon^{k-1}, \ldots, p^{k-1}\epsilon)$:

1. If $\deg f < 1$: Print $f(0)$ and stop.
2. Find $h \in B$ with $g^p - h \in pA$. Set $\beta \leftarrow (g + \epsilon)^p - h$.
3. Set $j \leftarrow p - 1$ and $s \leftarrow 0$.
4. Compute $f_j(h + \delta)$ in $B[\delta]/(\delta^k, p\delta^{k-1}, \ldots, p^{k-1}\delta)$ by Algorithm C recursively, where $f(z) = f_0(z^p) + zf_1(z^p) + \cdots + z^{p-1}f_{p-1}(z^p)$.
5. Set $s \leftarrow (g + \epsilon)s + \sum b_i\beta^i$, where $f_j(h + \delta) = \sum b_i\delta^i$.
6. If $j = 0$: Print s and stop.
7. Decrease j by 1 and return to step 4.

The idea of Algorithm C is as follows. Consider $f(g + pt)$ in the polynomial ring $A[t]$. It equals $f_0((g+pt)^p)+\cdots+(g+pt)^{p-1}f_{p-1}((g+pt)^p)$. To compute $f_j((g+pt)^p)$, find $h \in B$ with $g^p - h \in pA$, and find $v \in A[t]$ satisfying $h + pv = (g + pt)^p$; recursively compute $f_j(h + pu)$ in the polynomial ring $B[u]$; then substitute $u \mapsto v$ to obtain $f_j((g + pt)^p)$.

To avoid multiplications and divisions by p, Algorithm C works with polynomials in $\epsilon = pt$ and $\delta = pu$. Thus $f(g+pt)$ becomes $f(g+\epsilon)$, $f_j(h+pu)$ becomes $f_j(h+\delta)$, and $u \mapsto v$ becomes $\delta \mapsto (g + \epsilon)^p - h$.

Algorithm C computes $f(g + \epsilon)$, not merely $f(g)$. One can save some time by eliminating ϵ at the top level of the recursion, if the goal is to compute $f(g)$. The recursive call in step 4 still needs $f_j(h + \delta)$, not merely $f_j(h)$.

Details and improvements. I represent the ring $A[\epsilon]/(\epsilon^k, p\epsilon^{k-1}, \ldots, p^{k-1}\epsilon)$ by $A[\epsilon]/\epsilon^k$. To multiply in $A[\epsilon]/\epsilon^k$, I do $k(k + 1)/2$ multiplications in A. For large k there are faster algorithms; see, e.g., [3].

To compute $(g+\epsilon)^p$ in step 2, I perform $p-1$ multiplications by $g+\epsilon$ in $A[\epsilon]/\epsilon^k$. Each multiplication by $g+\epsilon$ is implemented with k multiplications in A. There are many faster algorithms; see, e.g., [5, section 4.6.3].

I choose h in step 2 as $\sum h_i y^i$ where h_i is the coefficient of x^{pi} in g^p. Then h_i can be extracted from $(g + \epsilon)^p$ with no extra arithmetic.

The sum $\sum b_i\beta^i = f_j(h + \beta)$ in step 5 can be viewed as an order-k composition over A. Horner's rule, which uses $k - 1$ multiplications in $A[\epsilon]/\epsilon^k$, suffices for small k. The first Brent-Kung algorithm is better for large k.

One can skip the multiplication of $g+\epsilon$ by s in step 5 when $j = p-1$, since $s = 0$. One can speed up some of the remaining multiplications by taking advantage of the sparseness of B inside A.

Speed. Let μ be a nondecreasing function such that elements of $R[x]/x^n$ can be multiplied in time $n\mu(n)$. I usually assume **fast multiplication**, meaning that $\mu(n) \in n^{o(1)}$ for $n \to \infty$. See [3] for a fast multiplication method. See [1] for a survey of multiplication methods.

Algorithm C's run time is dominated by multiplications. Its multiplication time is at most

$$c\left(e(n-1) + \frac{p^e - 1}{p-1}\right)\mu(n)$$

if $\deg f < p^e$, where $c = (2p-1)k + p(k-1)k(k+1)/2$. (Here c is the number of multiplications in A performed in steps 2 and 5 of Algorithm C. It can be reduced in several ways, as discussed above.) Indeed, for $\deg f < 1$, Algorithm C uses no multiplications. Otherwise it performs p recursive calls, taking time at most

$$pc\left((e-1)(m-1) + \frac{p^{e-1} - 1}{p-1}\right)\mu(m) \le c\left((e-1)(n-1) + \frac{p^e - p}{p-1}\right)\mu(n)$$

by induction, and c multiplications in A, taking time at most $cn\mu(n)$.

In particular, order-n power series composition takes time $O(n\mu(n)\log n)$ for fixed characteristic.

References

[1] Daniel J. Bernstein, *Multidigit multiplication for mathematicians*, preprint available from `http://pobox.com/~djb/papers/m3.dvi`.

[2] Richard P. Brent, H. T. Kung, *Fast algorithms for manipulating formal power series*, Journal of the ACM **25** (1978), 581–595.

[3] David G. Cantor, Erich Kaltofen, *On fast multiplication of polynomials over arbitrary algebras*, Acta Informatica **28** (1991), 693–701.

[4] Erich Kaltofen, Victor Shoup, *Subquadratic-time factoring of polynomials over finite fields*, preprint.

[5] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 2nd edition, Addison-Wesley, Reading, Massachusetts, 1981.

Department of Mathematics, Statistics, and Computer Science, The University of Illinois at Chicago, Chicago, IL 60607–7045

E-mail address: `djb@pobox.com`