ALGORITHMS FOR RATIONAL FUNCTION ARITHMETIC OPERATIONS

by

Ellis Horowitz

TR 71 - 112

November 19, 1971

(Revised August 1972)

Department of Computer Science
Cornell University
Ithaca, New York 14850

Algorithms for Rational Function Arithmetic Operations

by

Ellis Horowitz

Abstract

Despite recent advances in speeding up many arithmetic and algebraic algorithms plus an increased concern with algorithm analysis, no computing time study has ever been done for algorithms which perform the rational function arithmetic operations. Mathematical symbol manipulation systems which provide for operations on rational functions use algorithms which were initially given by P. Henrici in 1956. In this paper, these algorithms are precisely specified and their computing times analyzed. Then, new algorithms based on the use of modular arithmetic are developed and analyzed. It is shown that the computing time for adding and taking the derivative of univariate rational functions is 2 orders of magnitude faster using the modular algorithms. Also, the computing time for rational function multiplication will be one order of magnitude faster using the modular algorithm. The new method is generalized to the multivariate case and extensive empirical results are given.

CR Categories: 4.22, 5.10, 5.30;

Keywords: Rational functions, modular arithmetic, arithmetic operations, algebraic algorithms.

Algorithms for Rational Function Arithmetic Operations

1. Introduction

Recently work has dramatically progressed in developing efficient arithmetic and algebraic algorithms for a broad class of operations. For example, G. Collin's suggestion for computing multiplicative inverses in a finite field, [COL69a] and W.S. Brown's method for polynomial greatest common divisor calculation, [BRO71] are 2 such recently improved algorithms. At the same time many of these algorithms have been incorporated into computer systems which perform a variety of symbolic and algebraic calculations, e.g. ALTRAN, [HAL70] and SAC-1 [COL68]. Both of these systems restrict their operations to formulas for which there exists a canonical form, such as multivariate polynomials and rational functions. This permits the use of more efficient algorithms and hence increases the usefulness of these systems as a problem solving tool. Though analyses have been done for many of the sub-algorithms of these systems, no analyses has been done for the rational function arithmetic operations.

Both ALTRAN and SAC-1 allow for operations on multivariate rational functions. Also, symbol manipulation systems which allow for a more general data type still provide for rational function manipulation. Thus, efficient algorithms for performing the rational function arithmetic operations are critical to all of these systems. The methods that are currently used were originally described in J.ACM by P. Henrici, [HEN56]. He was concerned with rational number arithmetic and primarily with the question of how to avoid overflow during the computation steps. W.S. Brown in [BRO64] extended Henrici's methods to apply to rational functions as well as to rational numbers. Only in [COL68], pp.16, does G. Collins give a brief argument as to why the Henrici-Brown algorithms will be 2 to 4 times faster than conventional methods when applied to rational numbers.

Thus, this paper has 2 main goals. First, the Henrici-Brown algorithms for the rational function arithmetic operations will be specified and analyzed. The computing times for these algorithms will be derived, based on the latest, most efficient methods for polynomial arithmetic and greatest common divisor (gcd) calculation. The second goal will be to develop new methods based on the use of modular arithmetic. These new algorithms will be specified, their computing times analyzed and compared to the times for the Henrici-Brown algorithms.

2. Current Methods

Existing systems which allow for operations on rational functions, e.g. [HAL70], [COL68] and [MAN68] all use the algorithms which were initially described by Henrici in [HEN56], pp. 6-9. These algorithms were given for manipulating rational numbers with the idea of preventing overflow in the intermediate computations. These methods were directly generalized by W.S. Brown to apply to rational functions. In this section these algorithms will be presented and their computing times analyzed.

We first introduce some notation and give some preliminary results to aid in analyzing the later algorithms

Definition 2.1: Let $f(x_1,\ldots,x_n)$ and $g(x_1,\ldots,x_n)$ be real-valued functions defined on some common domain D. Then $f = O(g)$ if there exists a positive real number c such that $f(x_1,\ldots x_n) \leq c\cdot g(x_1,\ldots x_n)$ for all $(x_1,\ldots,x_n) \in D$.

The maximum computing times for operations on multi-precision integers will be given in terms of the maximum number of digits of the integers. Thus, given 2 integers bounded in magnitude by d, then the maximum computing time for adding or multiplying these integers is $O(\ln d)$ and $O((\ln d)^2)$ respectively. A new method for integer multiplication, due to S. Schonhage and V. Strassen [KNU 69], p.269 takes time $O((\ln d)(\ln \ln d)(\ln \ln \ln d))$. However, their method seems to be of more theoretical interest rather than practically useful and so we will assume the use of a classical multiplication algorithm.

Definition 2.2: Let $V(d,m) = \{P(x)\colon \deg(P) \leq m$ and all coefficients of $P(x)$ are bounded in magnitude by $d\}$.

Polynomials are assumed to have integer coefficients. Then, rational functions will be regarded as numerator-denominator pairs of relatively prime polynomials such that the leading coefficient (ldcf) of the denominator is positive. This defines a unique representation for rational functions which is necessary for precisely defining algorithms which manipulate these entities.

<u>Theorem 2.1</u>: Let $P(x)$, $Q(x)$, $P(x)/Q(x) \in V(d,n)$. Then the time to compute $P(x) \pm Q(x)$ is $O(n \ln d)$ and the time to compute $P(x) \cdot Q(x)$ or $P(x)/Q(x)$ is $O(n^2(\ln d)^2)$.

<u>Proof</u>: see [COL69b], pp. 206.

Now in all of the following algorithms to be developed, the operation of determining the greatest common divisor (gcd) of 2 polynomials will be critical.

<u>Theorem 2.2</u>: Let $P(x)$, $Q(x) \in V(d,n)$ and $A = \gcd(P,Q)$. Then if $k = \text{degree}(A)$, $A \in V((nd)^k, k)$ and the maximum computing time to find A is $O(n(n - k+1)^2(\ln d)^2)$.

<u>Proof</u>: see [COL69b], pp. 213

Thus, it is possible that the factors of a given polynomial can have much larger coefficients than the coefficients of the product. Since we will be interested in counting all multi-precision arithmetic operations, these bounds are extremely important.

We now begin to examine the Henrici-Brown algorithms. The analysis will first be done for rational functions in one variable This will be sufficient since even for the one variable case we will later develop a new algorithm which is considerably better. In a later section the generalization to multivariate rational functions will be made.

Algorithm RSUM:

Input: 2 rational functions R_1/R_2, S_1/S_2 in lowest terms.

Output: T_1/T_2 where $T_1/T_2=(R_1/R_2) + (S_1/S_2)$ a rational function in lowest terms;

1) $B \leftarrow \gcd(R_2, S_2)$;
2) $\bar{R}_2 \leftarrow R_2/B$; $\bar{S}_2 \leftarrow S_2/B$;
3) $\bar{T}_1 \leftarrow R_1\bar{S}_2 + S_1\bar{R}_2$;
4) $\bar{T}_2 \leftarrow R_2\bar{S}_2$;
5) $A \leftarrow \gcd(\bar{T}_1, B)$;
6) $T_1 \leftarrow \bar{T}_1/A$; $T_2 \leftarrow \bar{T}_2/A$;
7) end.

Theorem 2.3: Let R_1/R_2, S_1/S_2 be 2 given univariate rational functions such that $R_1, R_2, S_1, S_2 \in V(d,n)$. Then the maximum computing time for Algorithm RSUM is $O(n^5(\ln nd)^2)$.

Proof: Let $k_1 = \deg(B)$, $k_2 = \deg(A)$ in Algorithm RSUM. Then

Step	Computing Time	Comments
(1)	$n(n-k_1+1)^2(\ln d)^2$	$B \in V((nd)^{k_1}, k_1)$
(2)	$k_1^3(n-k_1)(\ln nd)^2$	$\bar{R}_2, \bar{S}_2 \in V((nd)^{k_1}, n-k_1)$
(3)	$nk_1(n-k_1)(\ln d)(\ln nd)$	$\bar{T}_1 \in V((nd)^{k_1}, 2n-k_1)$
(4)	$nk_1(n-k_1)(\ln d)(\ln nd)$	
(5)	$k_1^2(2n-k_1)(2n-k_1-k_2+1)^2(\ln nd)^2$	$A \in V((nd)^{k_1 k_2}, k_2)$
(6)	$k_1^2k_2^3(2n-k_1-k_2)(\ln nd)^2$	

It is easy to see that step (5) bounds the time for this algorithm. Since $B = \gcd(R_2, S_2)$, it follows that $k_2 \leq k_1 \leq n$. Letting $k_1 = k_2 = n$, we have that the maximum computing time is $O(n^5(\ln nd)^2)$.

<u>Algorithm RPROD</u>:

Input: 2 rational functions R_1/R_2, S_1/S_2 in lowest terms;

Output: $T_1/T_2 = (R_1/R_2)(S_1/S_2)$, a rational function in lowest terms.

1) $A \leftarrow \gcd(R_1, S_2)$;
2) $B \leftarrow \gcd(R_2, S_1)$;
3) $\bar{R}_1 \leftarrow R_1/A$; $\bar{S}_2 \leftarrow S_2/A$; $\bar{R}_2 \leftarrow R_2/B$; $\bar{S}_1 \leftarrow S_1/B$;
4) $T_1 \leftarrow \bar{R}_1\bar{S}_1$; $T_2 \leftarrow \bar{R}_2\bar{S}_2$;
5) end;

<u>Theorem 2.3:</u> Let R_1/R_2, S_1/S_2 be 2 given univariate rational functions such that $R_1, R_2, S_1, S_2 \in V(d,n)$. Then, the maximum computing time for Algorithm RPROD is $O(n^4(\ln nd)^2)$.

<u>Proof</u>: Let $k_1 = \text{degree}(A)$, $k_2 = \text{degree}(B)$ in Algorithm RPROD above. Then,

<u>Step</u>	<u>Computing Time</u>	<u>Comments</u>
(1)	$n(n-k_1+1)^2(\ln d)^2$	$A \in V((nd)^{k_1},k_1)$
(2)	$n(n-k_2+1)^2(\ln d)^2$	$B \in V((nd)^{k_2},k_2)$
(3)	$((n-k_1)k_1{}^3+(n-k_2)k_2{}^3)(\ln nd)^2$	$\bar{R}_1,\bar{S}_2 \in V((nd)^{k_1},n-k_1)$ $\bar{R}_2,\bar{S}_1 \in V((nd)^{k_2},n-k_2)$
(4)	$(n-k_1)(n-k_2)k_1k_2(\ln nd)^2$	

It follows from steps (1) and (2) that $k_1 \leq n$, $k_2 \leq n$. Letting $k_1 \cong k_2 \cong \frac{n}{2}$, step (3) bounds the computation time, namely $O(n^4(\ln nd)^2)$.

Thus, we see that the computing time for rational function addition is $O(n^5(\ln nd)^2)$ and $O(n^4(\ln nd)^2)$ for rational function multiplication. It follows immediately that the times for subtraction and division are the same as those for addition and multiplication respectively. For subtraction differs from addition by only a multiplicative constant and division merely requires us to invert the rational function and multiply. Any reasonable data structure for a rational function should allow us to invert the numerator and denominator in a constant amount of time.

Finally, let us look at the Henrici-Brown method for computing the derivative of a rational function.

<u>Algorithm RDERIV</u>:

<u>Input</u>: A rational function $R_1(x)/R_2(x)$; in lowest terms;

<u>Output</u>: A rational function T_1/T_2 where $T_1/T_2 = (R_1/R_2)'$.

1) $R_1' \leftarrow \mathrm{PDERIV}(R_1)$; $R_2' \leftarrow \mathrm{PDERIV}(R_2)$;
2) $G \leftarrow \gcd(R_2, R_2')$;
3) $\bar{R}_2 \leftarrow R_2/G$; $\bar{R}_2' \leftarrow R_2'/G$;
4) $\bar{T}_1 \leftarrow R_1'\bar{R}_2 - R_1\bar{R}_2'$;
5) $\bar{T}_2 \leftarrow R_2\bar{R}_2$;
6) $H \leftarrow \gcd(\bar{T}_1, G)$;
7) $T_1 \leftarrow \bar{T}_1/H$; $T_2 \leftarrow \bar{T}_2/H$
8) end.

<u>Theorem 2.4:</u> Let R_1/R_2 be a univariate rational function, R_1, $R_2 \in V(d,n)$. Then, the maximum computing time for Algorithm RDERIV is $O(n^5(\ln nd)^2)$.

<u>Proof</u>: Let k_1 = degree(G), k_2 = degree(H) in Algorithm RDERIV.

<u>Step</u>	<u>Computing Time</u>	<u>Comments</u>
(1)	$n(\ln n)(\ln d)$	$R_1', R_2' \in V(nd, n-1)$
(2)	$n(n-k_1+1)^2(\ln nd)^2$	$G \in V((nd)^{k_1}, k_1)$
(3)	$k_1^3(n-k_1)(\ln nd)^2$	$\bar{R}_2, \bar{R}_2' \in V((nd)^{k_1}, n-k_1)$
(4)	$(n-1)(n-k_1)k_1(\ln nd)^2$	$\bar{T}_1 \in V((nd)^{k_1+1}, 2n-k_1+1)$
(5)	$k_1(n-1)(n-k_1)(\ln nd)^2$	$\bar{T}_2 \in V((nd)^{k_1}, 2n-k_1)$
(6)	$k_1^2(2n-k_1+1)(2n-k_1-k_2+2)^2(\ln nd)^2$	$H \in V((nd)^{k_1k_2}, k_2)$
(7)	$k_2(2n-k_1-k_2+1)k_1^2k_2^2(\ln nd)^2$	

It follows from steps (2) and (6) that $k_1 \leq n-1$ and $k_2 \leq n-1$. Letting k_1, k_2 assume their maximum values, step (7) will bound the computation time which is $n(n^2)n^2(\ln nd)^2 = O(n^5(\ln nd)^2)$.

Thus we see that rational function addition and taking the derivative can take an order of magnitude more time than forming the product when using the Henrici-Brown algorithms.

3. Modular Methods

Now let us proceed to develop algorithms for rational function manipulation using modular arithmetic techniques. These techniques have already proved useful for speeding up such processes as polynomial gcd calculation [BRO71] and rational function integration [HOR70b]. A review of the use of modular arithmetic as applied to algebraic algorithms can be found in [HOR70a].

Briefly, let us restate some of the computing times for the basic operations. We assume a list of single precision primes $p_1,\ldots,p_k$ are available. Given 2 elements a, b in the finite field with p_i elements, $GF(p_i)$, then the time to compute $a+b$ or $a\cdot b$ is $O(1)$. If d is any integer, then the time to compute $d \bmod p_i$ is $O(\ln d)$.

In general, the use of modular arithmetic can be described as a 3 step process:

1) Given inputs d, e compute $(d_1 \bmod p_1,\ldots,d_k \bmod p_k)$ and $(e_1 \bmod p_1,\ldots,e_k \bmod p_k)$
2) Manipulate d and e using their radix representations producing a final k-tuple $(c_1 \bmod p_1,\ldots,c_k \bmod p_k)$
3) Apply the Chinese Remainder Algorithm to produce the unique c such that $c \equiv c_i \bmod p_i$, $1 \leq i \leq k$.

The use of Garner's method to implement the Chinese Remainder algorithm gives a computing time of $O(k^2)$, see Knuth [KNU69]. A recent paper by L. Heindel and E. Horowitz, [HEI71] has shown theoretically how this time could be reduced. However, at the moment their method is not practical and hence Garner's version will be assumed.

There are 2 other issues associated with the use of modular arithmetic. The first is deciding how many primes must be processed before the correct answer with integer coefficients can be obtained. One way to decide this is to compute an a priori bound on the size of the coefficients of the results using the inputs. This is, in fact, what will be done in the following algorithms. A way to possibly avoid this bound is to perform a quick check to see if the temporary answer is the correct one. However, the time for this test must be small to make it worthwhile. A second problem which may occur is when the leading coefficient of some polynomial becomes divisible by p_i and thus becomes zero. As Brown shows in [BRO71], pp. 43-44, the chance of a coefficient being divisible by a prime is extremely unlikely, the probability being approximately $1/p_i$. Since p_i is large (p_i is approximately $2^{32}-1$ on an IBM360) the probability of a rejected prime is very small. Thus, in practice an initial test for divisibility is made, plus a test after each prime has been processed.

Let us now examine a modular algorithm for adding 2 rational functions. All polynomials with coefficients in $GF(p_i)$ will be starred.

<u>Algorithm MRSUM</u>:

<u>Input</u>: 2 rational functions R_1/R_2, S_1/S_2 in lowest terms;

<u>Output</u>: $T_1/T_2 = (R_1/R_2) + (S_1/S_2)$, a rational function in lowest terms.

1) [Initialze]
$n_i \leftarrow \deg(R_i) \quad 1 \leq i \leq 2$
$m_i \leftarrow \deg(S_i)$
$n \leftarrow \max\{n_1+m_2,\ n_2+m_1\ n_2+m_2\}$
$d \leftarrow \max\{$magnitude of the coefficients of $R_1, R_2, S_1, S_2\}$;
$k \leftarrow 2d^n$; $i \leftarrow 0$; $q \leftarrow 1$;
$T_1 \leftarrow T_2 \leftarrow 1$;

2) [Next Prime]
$i \leftarrow i+1$; Choose next prime p_i. If none left, stop;

3) [Reject Prime]
If p_i divides either $\mathrm{ldcf}(R_1)$ or $\mathrm{ldcf}(R_2$ or $\mathrm{ldcf}(S_1)$ or $\mathrm{ldcf}(S_2)$, then go to (2);

4) [Apply Modular] [Homomorphism]
$R_j^* \leftarrow R_j \bmod p_i$, $S_j^* \leftarrow S_j \bmod p_i$, $1 \leq j \leq 2$

5) [Apply Modular] [RSUM Routine]
$T_1^* \leftarrow R_1^* S_2^* + R_2^* S_1^*$; $T_2 \leftarrow R_2^* S_2^*$;
$A^* \leftarrow \gcd(T_1^*, T_2^*)$;
$\overline{T}_1^* \leftarrow T_1^*/A^*$; $\overline{T}^* \leftarrow T_2^*/A^*$;

6) [Apply Chinese] [Remainder Alg]
If $\deg(\overline{T}_1^*) > \deg(T_1)$ or $\deg(\overline{T}_2^*) > \deg($
Then $T_1 \leftarrow \overline{T}_1^*$; $T_2 \leftarrow \overline{T}_2^*$;
$q \leftarrow p_i$; go to (2);)
If $\deg(\overline{T}_1^*) < \deg(T_1)$ or $\deg(\overline{T}_2^*) < \deg($
Then go to (2);
Find $\overline{T}_1, \overline{T}_2$: $\overline{T}_1 \equiv \overline{T}_1^* \bmod p_i$, $\overline{T}_2 \equiv \overline{T}_2^*$ mo
$\overline{T}_1 \equiv T_1 \bmod q$, $\overline{T}_2 \equiv T_2$ mod

7) [done?]
$T_1 \leftarrow \overline{T}_1$; $T_2 \leftarrow \overline{T}_2$;
$q \leftarrow q \cdot p_i$; If $q < k$, then go to (2)
end;

<u>Theorem 3.1</u>: Let R_1/R_2, S_1/S_2 be 2 univariate rational functions ~~such that $R_1, R_2, S_1, S_2 \in V(d,n)$. Then, the maximum computing~~ time for Algorithm MRSUM is $O(n^3(\ln d)^2)$.

Proof:

Step	Computing Time	Comments
(1)	$m(\ln d)$	
(2)	m	let m be the total number of primes processed plus rejected.
(3)	$m(\ln d)$	
(4)	$mn(\ln d)$	
(5)	$m(2n^2 + n^2 + n^2) = 4mn^2$	Both multiplication and gcd calculation of 2 n-degree polynomials takes $O(n^2)$ ste
(6)	nm^2	Using Garner's version of t Chinese Remainder Algorithm applied to each coefficient
(7)	$m(\ln d)^2$	

Thus, we see that $mn^2 + nm^2$ bounds the computing time. Assuming that $m = O(n \ln d)$ it follows that the time for MRSUM is $O(n^3(\ln d) + n^3(\ln d)^2) = O(n^3(\ln d)^2)$.

Algorithm MRPROD:

Input: 2 rational functions R_1/R_2, S_1/S_2 in lowest terms

Output: $T_1/T_2 = (R_1/R_2)(S_1/S_2)$, a rational function in lowest terms;

1) [Initialize] $n_i \leftarrow \deg(R_i)$, $m_i \leftarrow \deg(S_i)$ $1 \le i \le 2$; $n \leftarrow \max\{n_1+m_1, n_2+$

$d \leftarrow \max\{$magnitude of coefficients of $R_1, R_2, S_1, S_2\}$

$k \leftarrow 2d^n$;

$T_1 \leftarrow T_2 \leftarrow 1$

2) [Next Prime] $i \leftarrow i+1$; Choose next prime, p_i; If none left, stop.

3)	[Reject Prime]	If p_i divides ldcf(R_1) or ldcf(R_2) or ldcf(S_1) or ldcf(S_2) then, go to (2);
4)	[Apply Modular] [Homomorphism]	$R_j^* \leftarrow R_j$ mod p_i $S_j^* \leftarrow S_j$ mod p_i, $1 \leqslant j \leqslant 2$.
5)	[Apply Modular] [Multiplication]	$T_1^* \leftarrow R_1^* S_1^*$; $T_2^* \leftarrow R_2^* S_2^*$; $A^* \leftarrow$ gcd(T_1^*, T_2^*) $\bar{T}_1^* \leftarrow T_1^*/A^*$; $\bar{T}_2^* \leftarrow T_2^*/A^*$;
6)	[Apply Chinese] [Remainder Alg.]	If deg(T_1^*) < deg(T_1) or deg(T_2^*) < deg(T_2) then go to (2); If deg(T_1^*) > deg(T_1) or deg(T_2^*) > deg(T_2) then ($T_1 \leftarrow T_1^*$; $T_2 \leftarrow T_2^*$; $q \leftarrow p_i$; go to (2): Find $\overline{T}_1$, $\overline{T}_2$: $\overline{T}_1 \equiv \overline{T}_1^*$ mod p_i $\overline{T}_2 \equiv \overline{T}_2^*$ mod p_i $T_1 \equiv T_1$ mod q $T_2 \equiv T_2$ mod q;
7)	[end?]	$T_1 \leftarrow \overline{T}_1$; $T_2 \leftarrow \overline{T}_2$; $q \leftarrow q \cdot p_i$. If q < k, to to (2); else end.

<u>Theorem 3.2</u>: Let R_1/R_2, S_1/S_2 be 2 univariate polynomials such that R_1, R_2, S_1, $S_2 \in V(d,n)$. Then the maximum computing time for Algorithm MRPROD is $O(n^3(\ln d)^2)$.

<u>Proof</u>: Let m be the total number of primes processed.

Step	Computing Time	Comments
(1)	$n(\ln d)$	
(2)	m	
(3)	$m(\ln d)$	
(4)	$nm(\ln d)$	
(5)	$n^2 m$	
(6)	nm^2	q represents the product of all previously used primes.
(7)	$m(\ln d)^2$	

Thus, steps (4), (5) and (6) bound the computing time for MRPROD. Under the assumption that the number of rejected primes is small, $m = O(n(\ln d))$, the computing time is $O(n^2(\ln d)^2 + n^3(\ln d) + n^3(\ln d)^2) = O(n^3(\ln d)^2)$. Again we have the situation that subtraction and division are direct analogs of addition and multiplication.

Modular algorithms have been presented for performing rational function addition, subtraction, multiplication and division. Let us now examine a modular algorithm for finding the derivative of a rational function.

Algorithm MRDERIV:

Input: A rational function $R_1(x)/R_2(x)$;

Output: A rational function T_1/T_2 equal to the derivative with respect to x of R_1/R_2;

1)	[Initialize]	$n_i \leftarrow \deg(R_i)$ $1 \le i \le 2$; $n \leftarrow \max\{n_1+n_2-1, 2n_2\}$ $d \leftarrow \max\{$coefficients of R_1, R_2 in magnitud $k \leftarrow 2d^n$; $i \leftarrow 0$; $q \leftarrow 1$; $T_1 \leftarrow T_2 \leftarrow 1$;
2)	[Next Prime]	$i \leftarrow i+1$; Choose next prime, p_i. If none left, end.
3)	[Reject Prime]	If p_i divides $\mathrm{ldcf}(R_1)$ or $\mathrm{ldcf}(R_2)$, go to (
4)	[Apply Modular] [Homomorphism]	$R_1^* \leftarrow R_1 \bmod p_i$; $R_2^* \leftarrow R_2 \bmod p_i$;
5)	[Apply Modular] [Derivative Alg.]	$T_1^* \leftarrow R_1^{*\prime} R_2^* - R_1^* R_2^{*\prime}$; $T_2^* \leftarrow R_2^{*2}$; $A^* \leftarrow \gcd(T_1^*, T_2^*)$; $\bar{T}_1^* \leftarrow T_1^*/A^*$; $\bar{T}_2^* \leftarrow T_2^*/A^*$;

6) [Apply Chinese]
[Remainder Alg.] If $\deg(\overline{T}_1^*) < \deg(T_1)$ or $\deg(\overline{T}_2^*) < \deg(T_2$
then go to (2);
If $\deg(\overline{T}_1^*) > \deg(T_1)$ or $\deg(\overline{T}_2^*) > \deg(T_2$
then $(T_1 \leftarrow \overline{T}_1^*;\ T_2 \leftarrow T_2^*;\ q \leftarrow p_i;$ go to (2)
Find $T_1, \overline{T}_2$: $\overline{T}_1 \equiv \overline{T}_1^* \bmod p_i,\ T_2 \equiv T_2^* \bmod$
$T_1 \equiv T_1 \bmod q, \quad T_2 \equiv T_2 \bmod q$

7) [end?] $T_1 \leftarrow T_1;\ \overline{T}_2 \leftarrow T_2;$
$q \leftarrow q \cdot p_i$
If $q < k$, go to (2);
else end.

<u>Theorem 3.3</u>: Let R_1/R_2 be a rational function with $R_1, R_2 \in V(d,n$
Then the maximum computing time for Algorithm MRDERIV(R_1/R_2) is $O(n^3(\ln d)^2)$.

<u>Proof</u>: Let m be the number of primes processed.

<u>Step</u>	<u>Computing Time</u>	<u>Comments</u>
(1)	$n(\ln d)$	
(2)	m	
(3)	$m(\ln d)$	test if p_i divides leading coefficient;
(4)	$nm(\ln d)$	
(5)	n^2m	
(6)	nm^2	q is the product of used primes;
(7)	m	

Thus steps (4), (5) and (6) bound the computation time. Since $m = O(n \ln d)$ it follows that the total computing time is bounded by $O(n^2(\ln d)^2 + n^3(\ln d) + n^3(\ln d)^2) = O(n^3(\ln d)^2)$.

4. Multivariate Rational Operations

In Section 3 we have seen that a modular based algorithm for rational function addition will be 2 orders of magnitude faster than the Henrici-Brown algorithm. In this section this fast modular method will be generalized to rational functions in several variables. The basic technique will be to apply the modular homomorphism to $R(x_1,\ldots,x_r)$, $S(x_1,\ldots,x_r)$ producing rational functions with coefficients in some finite field. Then the algorithms will evaluate R* and S* at many points for each variable producing a collection of univariate rational functions. These rational functions will be added and the results interpolated until the correct $T^*(x_1,\ldots,x_r)$ is produced such that $T^* = R^* + S^*$. This technique has already been employed to advantage by W.S. Brown [BRO71] and others to speed up multivariate algebraic operations such as gcd calculation and it is now a standard technique for performing multivariate operations. Thus in this section I will show how the technique works in addition.

Let us now recall the computing times for performing certain basic operations on polynomials.

<u>Definition 4.1</u>: Let $M_r(d,m) = \{P(x_1,\ldots,x_r) : (\deg(P) \text{ in } x_i+1) \leq$ for $1 \leq i \leq r$ and every coefficient of P is bounded by d$\}$.

Thus if $P(x_1,\ldots,x_r) \in M_r(d,m)$ then P contains at most m^r terms.

<u>Theorem 4.1</u>: Let $P(x_1,\ldots,x_r)$, $Q(x_1,\ldots,x_r) \in M_r(d,m)$. Then an upper bound on the computing time to form $P \pm Q$ and $P \cdot Q$ is $O(m^r(\ln d))$ and $O(m^{2r}(\ln d)^2)$ respectively.

We note that if instead P and Q had coefficients in GF(p), p single precision then $P,Q \in M_r(1,m)$ and the time for addition and multiplication is $O(m^r)$ and $O(m^{2r})$.

There are some other operations for which we will need the computing times. In particular we need to know the times for applying modular and evaluation homomorphisms to polynomials and for applying interpolation and the Chinese Remainder method to integers and elements of GF(p). These methods have already been fully documented in, for example, [BRO71], pp. 496-500. Let us review those results here.

<u>Theorem 4.2</u>: Let $A \in M_r(d,m)$ such that $A = \sum_{1 \leq i \leq m^r} a_i x_1^{e_{i1}} \cdots x_r^{e_{ir}}$ and p a single precision prime. Then an upper bound on the time to compute $A^* = \sum_{1 \leq i \leq m^r} (a_i \bmod p)\, x_1^{e_{i1}} \cdots x_r^{e_{ir}}$ is $O(m^r \ln d)$.

<u>Proof</u>: There are at most m^r non-zero terms, each term requiring a division by a single precision prime p. This implies $O(\ln d)$ for each term.

This operation is referred to as a modular homomorphism since we are mapping the polynomials in $I[x_1,\ldots,x_r] \to GF(p)[x_1,\ldots,x_r]$ and this mapping satisfies the properties of a homomorphism.

<u>Theorem 4.3</u>: Let $A^*(x_1,\ldots,x_r) \in GF(p)[x_1,\ldots,x_r]$ and suppose $b \in GF(p)$. Then an upper bound on the time to form $A^*(x_1,\ldots,x_{r-1},b)$ is $O(m^r)$.

<u>Proof</u>: [BRO71], pp. 496-500.

We shall call this an <u>evaluation homomorphism</u> since we are mapping from $GF(p)[x_1,\ldots,x_r] \to GF(p)[x_1,\ldots,x_{r-1}]$ and this mapping also satisfies the homomorphism property.

Now let us investigate the inverse operations of interpolation and the Chinese Remainder algorithm. The analogies between these two operations have already been mentioned before; namely that the Chinese Remainder algorithm allows us to construct an integer from its images modulo several primes while interpolation allows us to construct a polynomial from its images modulo $x-b_1,\ldots,x-b_n$. We now present these two methods as they would be applied to multivariate polynomials.

Iterative Chinese Remainder Algorithm

Input: $A_1(x_1,\ldots,x_r)$ with coefficients in GF(p)

$A_2(x_1,\ldots,x_r)$ with coefficients in GF(q)

Output: $A_3(x_1,\ldots,x_4)$: $A_3 \equiv A_1 \bmod p$

$A_3 \equiv A_2 \bmod q$

1) $A_2^* \leftarrow A_2 \bmod p$;

2) $q^* \leftarrow q \bmod p$;

3) $C \leftarrow (A_1 - A_2^*)/q^*$;

4) $A_3 \leftarrow Cq + A_2$;

Theorem 4.4: An upper bound on the computing time for one iteration of the Chinese Remainder Algorithm given above is $O(m^r \ln q)$.

Proof: The times for steps (1) and (4) are bounded by $m^r(\ln q)$ whereas step (2) takes no more than $O(\ln q)$ and step (3) takes $O(m^r)$.

Iterative Interpolation

Input: $A_1(x_1,\ldots,x_{r-1})$ with coefficients in GF(p);

$b \in GF(p)$;

$A_2(x_1,\ldots,x_r)$ with coefficients in GF(p);

$Q(x_r)$, degree$(Q) \leq k$, $Q = \prod_{1\leq i\leq k} (x-b_i)$

Output: $A_3(x_1,\ldots,x_r)$: $A_3(x_1,\ldots,x_{r-1},b_i) = A_2(x_1,\ldots,x_{r-1},b_i)$

and $A_3(x_1,\ldots,x_{r-1},b) = A_1(x_1,\ldots,x_{r-1})$

1) $A_2^* \rightarrow A_2(x_1,\ldots,x_{r-1},b)$;

2) $q^* \rightarrow Q(b)$;

3) $C \rightarrow (A_1-A_2^*)/q^*$;

4) $A_3 \rightarrow CQ + A_2$;

Theorem 4.5: An upper bound on the computing time for one iteration of the interpolation algorithm above is $O((m^{r-1})k)$.

Proof: The time for steps (1), (3), and (4) is $O((m^{r-1})k)$. The time for step (2) is $O(m)$.

We are now in a position to present and analyze an algorithm for multivariate rational function addition. Specifically, 2 algorithms will be presented CRSUM and a modified MRSUM where CRS is a subroutine of the latter.

Algorithm CRSUM (p,R^*,S^*)

Input: $R^*(x_1,\ldots,x_r) = R_1^*/R_2^*$, $S^*(x_1,\ldots,x_r) = S_1^*/S_2^*$

$R_1^*,R_2^*,S_1^*,S_2^* \in GF(p)[x_1,\ldots,x_r]$, $r \geq 1$;

Output: $T^*(x_1,\ldots,x_r) = R^* + S^*$, a rational function in lowest terms, $T^* \in GF(p)[x_1,\ldots,x_r]$;

1) [One variable] If $r = 1$ then $(\overline{T}_1 \leftarrow R_1^*S_2^* + R_2^*S_1^*$;

$\overline{T}_2 \leftarrow R_2^*S_2^*$; $A \leftarrow \gcd(\overline{T}_1,T_2)$

$T_1^* \leftarrow \overline{T}_1/A$; $T_2^* \leftarrow \overline{T}_2/A$;

end;)

2) [Initialize] $n_{ij} \leftarrow \deg(R_i^*)$ in x_j, $m_{ij} \leftarrow \deg(s_i)$ in x_j

$1\leq i\leq 2$, $r-1\leq j\leq r$; $C_1 \leftarrow C_2 \leftarrow 0$; $D(x_r) \leftarrow 1$;

$b \leftarrow -1$; $n \leftarrow \max\{m_{2,r}+n_{2,r}, m_{1,r}+n_{2,r}, m_{2r}+n_{1}$

4) [Evaluate] $\overline{R}_i \leftarrow R_i^*(x_1,\ldots,x_{r-1},b)$,

$\overline{S}_i \leftarrow S_i^*(x_1,\ldots,x_{r-1},b)$ $1\underline{<}i\underline{<}2$;

If $\deg(\overline{R}_i)$ in $x_{r-1} < n_{i,r-1}$ or

$\deg(\overline{S}_i)$ in $x_{r-1} < m_{i,r-1}$ for $1\underline{<}i\underline{<}2$

then go to (3);

5) [Recursion] $\overline{T}(x_1,\ldots,x_{r-1}) \leftarrow$ CRSUM $(p,\overline{R}_1/\overline{R}_2,\overline{S}_1/\overline{S}_2)$;

6) [Interpolate] $C_i(x_1,\ldots,x_4) \leftarrow \dfrac{\overline{T}_i - C_i(x_1,\ldots,x_{r-1},b)}{D(b)} \cdot D($

$+ C_i$ for $1\underline{<}i\underline{<}2$.

7) [Done] $D(x_r) \underline{<} (x-b)D(x_r)$;

If $\deg(D) \underline{<} n$ then go to (3);

8) [end] $T_1^* \leftarrow C_1$; $T_2^* \leftarrow C_2$; $T \leftarrow T_1^*/T_2^*$; end.

Now let us analyze the computing time for CRSUM.

<u>Theorem 4.5</u>: Let $R^*(x_1,\ldots,x_r)$, $S^*(x_1,\ldots,x_r)$ be rational functions in $r \geq 1$ variables with coefficients in GF(p). Let $R^* = R_1^*/S_1^*$, $S^* = S_1^*/S_2^*$ where $R_1^*,R_2^*,S_1^*,S_2^* \in M_r(1,m)$. Then the maximum computing time for algorithm CRSUM is $O(m^{2r})$.

<u>Proof</u>: If $r = 1$ then the times for the polynomial multiplications and gcd operations is $O(m^{2r}) = O(m^2)$. Step (2) is executed only once and requires only $O(m^r)$ operations to determine the degrees. Then steps (3) - (7) are executed at most 2m times. We then have the following table:

Step	Time	
(3)	$2m$	
(4)	$2m \cdot m^r$	by Theorem 4.3
(5)	$2mT(r-1)$	where $T(r)$ is the time for CRSUM1 with r-variable inpu
(6)	$\sum_{1 \le i \le 2m} m^{r-1} \cdot i \le 4m^{r+1}$	by Theorem 4.4.
(7)	$4m^2$	

Then from the table we see that steps (4), (5), and (6) bound the computing time, that is

$$\begin{aligned} T(r) &= 4m^{r+1} + 2mT(r-1) < m^{r+2} + m^2T(r-1) \\ &= m^{r+2} + m^{r+3} + m^4T(r-2) = \cdots = \\ &\quad m^r(m^2+m^3+\cdots+m^r) + m^{2(r-1)}T(1) \\ &= O(m^{2r}) . \end{aligned}$$

Now that we have analyzed CRSUM we can give an algorithm for adding multivariate rational functions with integer coefficients. In fact, algorithm MRSUM as given in Section 3 can be used with only a few changes.

1) Interpret R_1/R_2 and S_1/S_2 as rational functions in r-variables with integer coefficients;
2) Let n be the maximum degree occurring in all variables of R_1, R_2, S_1 or S_2;
3) Replace step (5) by $T^* \leftarrow \mathrm{CRSUM}(p, R_1^*/R_2^*, S_1^*/S_2^*)$;

Thus the application of the modular homomorphism and of the Chinese Remainder algorithm is made to every numerical coefficien of R_1, R_2, S_1, S_2 and $\overline{T}_1, \overline{T}_2$. Let us now reanalyze the time for MRSUM whose inputs are multivariate rational functions.

<u>Theorem 4.6</u>: Let $R(x_1, \ldots, x_r)$, $S(x_1, \ldots, x_r)$ be 2 rational functions in $r \geq 1$ variables, $R = R_1/R_2$, $S = S_1/S_2$ and $R_1, R_2, S_1, S_2 \in M_r(d,m)$. Then the maximum computing time for MRSUM(R,S) is $O(m^{2r+1}(\ln m^r d)^2)$.

<u>Proof</u>: The time to obtain the maximum degree in R_1, R_2, S_1, S_2 can require no more than $O(rm^r)$ operations. Then steps (2) - (7) will be executed $O(m \ln m^r d)$ times. We can then derive the following table:

<u>Step</u>	<u>Time</u>	
(2)	$m(\ln m^r d)$	
(3)	$m(\ln m^r d)(\ln d)$	
(4)	$m(\ln m^r d) \cdot m^r \ln d$	by Theorem 4.2
(5)	$m(\ln m^r d) \cdot m^{2r}$	by Theorem 4.5
(6)	$(m(\ln m^r d))^2 \cdot m^r$	by Theorem 4.4
(7)	$m^2(\ln m^r d)^2$	

Thus we see that steps (5) and (6) bound the computing time, namely $m^{r+2}(\ln m^r d)^2 + m^{2r+1}(\ln m^r d) = O(m^{2r+1}(\ln m^r d)^2)$.

Now recalling the computing time analysis in Section 3 for univariate rational functions, $r = 1$ and Theorem 4.6 yields $O(m^3(\ln md)^2)$ which is the same result derived there. We note that the multivariate algorithm for rational function product will have the same form and the same computing time analysis.

5. Empirical Results

In this section the results of some empirical studies will be presented to see if we can determine just how practical the new homomorphism algorithms really are. All testing was done on an IBM 360/65 at Cornell University. The basic routines for the manipulation of multivariate polynomials were provided through the use of the SAC-1 system for symbolic and algebraic calculation as described in [COL68]. This is a collection of Fortran subroutines which allows in part for multivariate polynomial and rat onal function arithmetic where the coefficients can be either in GF(p) or multi-precision integers.

In particular the Henrici-Brown algorithms for addition and multiplication were compared against MRSUM and MRPROD with rational function entries in one, two and three variables. The coefficients were randomly generated in the range $[-2^{10},2^{10}]$. Tables I, II and III give the resulting computing times in milliseconds. If R(x,y) is a rational function of total degree k, then x and y appear in the numerator and denominator of R to degree k or less.

TABLE I

UNIVARIATE RATIONAL FUNCTIONS

Methods	degrees 5	6	7	8	9	10
RSUM	432	716	915	1131	1565	1847
MRSUM	282	366	466	766	782	1032
RPROD	798	1182	1647	2412	3511	4792
MRPROD	310	390	480	623	724	832

TABLE II

BIVARIATE RATIONAL FUNCTIONS

Methods	total degree 2	3	4	5
RSUM	1681	7987	38638	266,832
MRSUM	632	1165	3760	6024
RPROD	2962	15985	120186	>300,000
MRPROD	528	1262	3284	7581

TABLE III

TRIVARIATE RATIONAL FUNCTIONS

Methods	total degree 2	3	4
RSUM	26391	131820	>300000
MRSUM	6520	31560	157827
RPROD	27677	149360	>300000
MRPROD	5832	27650	126387

6. Even Lower Bounds

In Sections 2 and 3 a basic fact about the size of the factors of a polynomial was used; namely, if $A(x) = \sum_{0 \leq i \leq n} a_i x_i$ $n = \deg(A)$ and $e \geq \ln(\sum_{0 \leq i \leq n} |a_i|)$ (i.e. where e bounds the precision of A's coefficients) and if B is a factor of A, then e bounds the precision of coefficients of B. In order to obtain upper bounds on the algorithms it is necessary to assume such coefficient growth. But for many cases this bound is highly unrealistic since if we divide a polynomial with e-precision coefficients by another we would normally expect a quotient whose coefficients are no more than e precision. In [BRO71], p. 497, Brown makes this assumption and argues that the computing times derived will then more closely reflect the behavior of the algorithms rather than be an upper bound which may grossly overestimate the algorithm's efficiency. In this section I will rederive the computing times for addition, multiplication and differentiation using his assumption. The I will show that the lowest asymptotic bounds on the rational function operations are obtainable by using the Schonhage-Strassen method for fast integer multiplication, [KNU69], p. 269 (2^{nd} edition) and the Schonhage algorithm for integer g.c.d. calculation [SCH72].

Incorporating Brown's assumption into the computing time analyses of the modular algorithms is straightforward since its only influence will be on m, the number of primes that are to be processed. Whereas we previously took m = O(n ln d) we can now assume that $m = O(\ln d)$. In all these algorithms, MRSUM, MRPROD, and MRDERIV the bounding expression is $n^2m + nm^2$ which now becomes $O(n^2 \ln d + n(\ln d)^2)$. We note that if one carries through the analysis on the Henrici-Brown algorithms using the same assumption, then they are still worse than the modular algorithms. For example, the time for RPROD would be $O(n^2(\ln d)^2) > O(n^2(\text{lnd } d) + n(\ln d)^2)$. Now the termination test in the modular algorithms must be altered to capitalize on the belief that often less than n ln d primes need to be processed. This can be accomplished as for instance in MRPROD by testing at step (7) if $T_1R_2S_2 = T_2R_1S_1$. The time for this test can be no worse than $O(n^2(\ln d)^2)$ times m and m will generally be $O(\ln d)$. Thus we still retain a slightly better bound than the modular algorithm analyzed under the more restrictive assumptions. Furthermore the algorithm will produce the result as soon as a sufficient numver of primes have been processed.

Finally we ask what is theoretically the best upper bounds we can obtain for these rational operations. For this let us first define the function $M(n,e) = ne \log ne(\log\log n3)$. In [KNU71], p. 269 ($2^{nd}$ edition) Knuth gives the fastest known method for e-precision integer multiplication. This algorithm takes time $M(1,e)$. The method can be directly generalized to the multiplication of n-degree, e-precision polynomials in time $M(n,e)$ simply by considering their coefficients as a large integer and inserting a sufficient number of zeros to prevent carries . Secondly, in a recent paper by Schonhage [SCH72] he gives a method for integer g.c.d. calculation which takes time O((log e)M(1,e) on e-precision numbers. Again the generalization to polynomials gives the bound O((log ne)M(n,e)). Using these two methods and analyzing the classical algorithm for rational function multiplication we find that O(nM(n,e)) bounds its computing time. Even recent work on decreasing the computing time for modular arithmetic, e.g. [HEI71] cannot produce as good a bound.

Thus in summary we have the following results:

Rational Operation	Henrici-Brown Algorithms	Modular Algorithms (upper bound)	Modular Algorit (average tim
addition	n^5e^2	n^3e^2	$n^2e + ne^2$
multiplication	n^4e^2	n^3e^2	$n^2e + ne^2$
differentiation	n^5e^2	n^3e^2	$n^2e + ne^2$

and the fastest asymptotic computing time being $nM(n,e) = O(n^2e \log ne(\log \log ne))$. It is unfortunate that these new, fast methods have such prohibitively large constants so that they seem infeasible to use in actual practice.

REFERENCES

[BRO64] Brown, W.S., J.P. Hyde and B.A. Tague, The ALPAK for Algebra on a Digital Computer-II: Rational Functions of Several Variables and Truncated Power Series with Rational Function Coefficients, B.S.T.J., Vol.43, No.1 (March 1964) pp. 785-804.

[BRO71] Brown, W.S., On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, J.A.C.M., Vol.18, No.4, October 1971.

[COL68] Collins, G.E., The SAC-1 Rational Function System, University of Wisconsin, Computing Center, Technical Report No.8 (July 1968) 31 pages.

[COL69a] Collins, G.E., Computing Multiplicative Inverse in GF(p), Math of Comp., Vol.23, No.105 (January 1969) pp. 197-200.

[COL69b] Collins, G.E., Computing Time Analyses for Some Arithmetic and Algebraic Algorithms, Proceedings of the 1968 Summer Institute on Symbolic and Mathematical Computation, (R.G. Tobey ed.) June 1969, IBM Federal Systems Center, pp. 195-231.

[HAL70] Hall, A.D., Jr., The ALTRAN System for Rational Function Manipulation, Los Angeles, March 1970, pp. 153-157.

[HEI71] Heindel, L. and E. Horowitz., On Decreasing the Computing Time for Modular Arithmetic, Proceedings - 12th Annual Symposium on Switching and Automata Theory, IEEE, East Lansing, Michigan, October 1971, pp. 126-128.

[HEN56] Henrici, P., A Subroutine for Computations with Rational Numbers, J. ACM, Vol.3, No.1 (January 1956) pp. 6-9.

[HOR70a] Horowitz, E., Modular Arithmetic and Finite Field Theory: A Tutorial, Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, March 1970, pp. 188-194.

[HOR70b] Horowitz, E., Algorithms for Partial Fraction Decomposition and Rational Function Integration, Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, March 1970, pp. 441-447.

[KNU69] Knuth, D.E., The Art of Computer Programming, Vol.2, Seminumerical Algorithms, Addison Wesley, 1969, 2nd edti

[MAN68] Manove, M., S. Bloom and C. Engelman, Rational Functions in MATHLAB, In Bobrow (ed.) Symbol Manipulation Languages and Techniques, North Holland, Amsterdam, 1968, pp. 86-102.

[SCH72] Schonhage, A., Fast Computation of Continued Fraction Expansions, (to appear).