# ON THE SUBSTITUTION OF POLYNOMIAL FORMS

Ellis Horowitz

TR 73 - 160

January 1973

Department of Computer Science
Cornell University
Ithaca, New York 14850

# On the Substitution of Polynomial Forms

Ellis Horowitz

## Abstract

The problem of devising efficient algorithms for computing $Q(x_1,\ldots,x_{r-1},P(x_1,\ldots,x_{r-1}))$ where P and Q are multivariate polynomials is considered. It is shown that for polynomials which are completely dense an algorithm based upon evaluation and interpolation is more efficient than Horner's method. Then various characterizations for sparse polynomials are made and the subsequent methods are re-analyzed. In conclusion a test is devised which takes only linear time to compute and by which a decision can automatically be made concerning whether to use a substitution algorithm which exploits sparsity or one which assumes relatively dense inputs. This choice yields the method which takes the fewest arithmetic operations.

# On the Substitution of Polynomial Forms

Ellis Horowitz

## Introduction

Recently there has been great interest in the operation of polynomial evaluation. For the case of computing $Q(x_0)$ where $x_0$ and the coefficients of Q are all single precision numbers, Horner's rule has been shown to provide the uniquely optimal method, see Borodin in [ 1 ]. Other work in reducing the required number of multiplications has been accomplished when one can assume that the coefficients of Q are known in advance. In this paper I seek to study the more general problem of substituting a multivariate polynomial, say $P(x_1,\ldots,x_{r-1})$ for the r-th variable in $Q(x_1,\ldots,x_r)$ thereby computing the unique polynomial in r-1 variables $Q(x_1,\ldots,x_{r-1},P(x_1,\ldots,x_{r-1}))$.

This operation is an important one in mathematical symbol manipulation systems. Computer software systems which provide for operations on symbolic mathematical formulas have been available for many years, e.g. see [9]. All of these systems allow for the handling of multivariate polynomials and most provide a substitution capability. Some of these systems, e.g. SAC-1 [4] and ALTRAN [6], use what is essentially a direct generalization of Horner's method to polynomials in several variables. It is the purpose of this paper to show that for several different, common classes of polynomials other methods will be as good or far superior.

Especially one should note that polynomial substitution can be only one of several important operations when symbolic manipulation systems are employed by such scientists as mathematicians. Often the cost of computing some expression can be far cheaper than the time for evaluating that expression. Thus it is necessary to carefully consider alternate strategies for performing effectively this task. In this paper we will see how the substitution operation can be quite costly and how other strategies can compare quite favorably to the classical scheme.

## 2. Assumptions

Before we begin an analysis of the substitution algorithms let us more precisely specify the types of polynomials we are dealing with and the computing times for basic operations.

We will be considering polynomials $P(x_1,\ldots,x_r)$ in $r \geq 1$ variables whose coefficients can be either:

(a) single precision fixed point numbers,

(b) single precision floating point numbers,

or

(c) elements of a finite field with a prime, p number of elements designated, GF(p).

We note that the arithmetic operations on all of these elements are bounded by a constant. The basic unit operation will be a single multiplication of 2 of these coefficients and all computing times will be a function of this unit. Of course we implicitly assert that the total number of additions-subtractions will be bounded by the number of multiplications.

All computing times will be given in terms of the commonly used O-notation.

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

Definition 2.2: Let $M(r,m) = \{P(x_1,\ldots,x_r)$: P has $r \geq 1$ variables with single precision coefficients such that $\deg(P)$ in $x_i \leq m$ for $1 \leq i \leq r\}$.

We now state without proof the computing times for the arithmetic operations on elements in $M(r,m)$.

Theorem 2.1 Let $P(x_1,\ldots,x_r)$ and $Q(x_1,\ldots,x_r) \in M(r,m)$. Then an upper bound on the computing time to form $P \pm Q$ is $O((m+1)^r)$ and to form $P \cdot Q$ is $O((m+1)^{2r})$.

Proof see e.g. [4]

These times represent the classical algorithms for addition and multiplication and reflect the methods that are commonly used in these systems today. There is hope that faster methods may soon be adopted for polynomial multiplication but at the moment we will assume the classical algorithms.

There are some other operations for which we will need the computing times. In particular we need to know the times for evaluation and interpolation of multivariate polynomials.

<u>Theorem 2.2</u> Let $P(x_1,\ldots,x_r) \in M(r,m)$ and suppose that $b \in GF(p)$. The time needed to compute $P(x_1,\ldots,x_{r-1},b)$ is $O((m+1)^r)$.

<u>Proof</u> P can be expressed in the form $P = \sum_{0\le i\le m} a_i(x_1,\ldots,x_{r-1}) \cdot x_r^i$. Computing $b^i$ for $0 \le i \le m$ takes m-1 multiplications. The $a_i$ each have $(m+1)^{r-1}$ terms so the total time is $m(m+1)^{r-1} + m - 1 = O((m+1)^r)$.

Next we will need to use an iterative method for interpolation. Thus I first give the algorithm

<u>Algorithm INTERPOLATION</u>

Input: $P(x_1,\ldots,x_{r-1}) \in M(r-1,m)$

$b \in GF(p)$

$Q(x_1,\ldots,x_r) \in M(r,m)$ except $\deg(Q)$ in $x_r = k$

$D(x_r)$ : $\deg(D) = k$

Output: $R(x_1,\ldots,x_r)$ such that

$R(x_1,\ldots,x_{r-1},b) = P(x_1,\ldots,x_{r-1})$

$R(x_1,\ldots,x_{r-1},b_i) = Q(x_1,\ldots,x_{r-1},b_i)$ where

$b_i$ are the roots of D.

```
1) $\overline{Q}(x_1,\ldots,x_{r-1}) \leftarrow Q(x_1,\ldots,x_{r-1},b)$;
2)       c          $\leftarrow D(b)$;
3) $H(x_1,\ldots,x_{r-1}) \leftarrow (P - \overline{Q})/c$;
4) $R(x_1,\ldots,x_r)$     $\leftarrow H \cdot D + Q$;
```

Theorem 2.3 An upper bound on the computing time for one execution of algorithm Interpolation is $O((m+1)^{r-1}k)$.

Proof The time for step (1) is $O((m+1)^{r-1}k)$ by Theorem 2.2. Step (2) is $O(m+1)$ using Horner's rule. Since P and $\overline{Q}$ have $(m+1)^{r-1}$ terms step (3) takes no more than $O((m+1)^{r-1})$ operations. Then step (4) takes $O((m+1)^{r-1}k)$ operations since D has degree k.

## 3. Dense Model

We are now in a position to analyze 3 methods for polynomial substitution. Two of these methods are well known while the third (SUBST) is newly presented. In this section the analysis will be carried out assuming that all the inputs are completely dense polynomials. In this case the assumption is that if $Q(x_1,\ldots,x_r) \in M(r,m)$ then all possible $(m+1)^r$ possible terms occur. Moreover as powers of Q are computed, these polynomials also remain completely dense.

The first method is due to Horner and generalized for its use with multivariate polynomials.

<u>Algorithm HORNER(P,Q)</u>

```
    Input:  $P(x_1,\ldots,x_{r-1})$, $Q(x_1,\ldots,x_r) = \sum_{0\le i\le m_r} a_i(x_1,\ldots,x_{r-1}) \cdot x_r^i$, $r \ge 1$, $m_r \ge 0$.

    Output: $R(x_1,\ldots,x_{r-1}) = Q(x_1,\ldots,x_{r-1},P)$.

1) [Initialize]   $R \leftarrow a_{m_r}$;

2) [Loop]         For $i \leftarrow m_r-1,\ldots,0$ do
                      $R \leftarrow R \cdot P + a_i$;

3) [End]          Return (R);
```

<u>Theorem 3.1</u> Let $P(x_1,\ldots,x_{r-1})$ and $Q(x_1,\ldots,x_r)$ be dense polynomials in $r \ge 1$ variables with single precision coefficients. Further suppose $n = \deg(P)$ in $x_i$, $1 \le i \le r-1$ and $m = \deg(Q)$ in $x_i$, $1 \le i \le r$. Then the computing time for HORNER(P,Q) is $O(m^r(n+1)^{2(r-1)})$.

<u>Proof</u> Since P and Q are dense P has $(n+1)^{r-1}$ terms and Q has $(m+1)^r$ terms. In step (2) of the algorithm, at the first iteration R has $(m+1)^{r-1}$ terms, at the second iteration R has $(m+n+1)^{r-1}$ terms, at the third iteration R has $(m+2n+1)^{r-1}$ terms and at the i-th iteration R has $(m+(i-1)n+1)^{r-1}$ terms for $1 \le i \le m$. Thus the time for step (2) is

$$\sum_{1\le i\le m} (m+(i-1)n+1)^{r-1}(n+1)^{r-1} \qquad (3.1)$$

$$= (n+1)^{r-1} \sum_{1\le i\le m} (m+(i-1)n+1)^{r-1}$$

$$< (n+1)^{r-1} \sum_{1\le i\le m} (m+(m-1)n+1)^{r-1}$$

$$= m(n+1)^{r-1}(m+(m-1)n+1)^{r-1}$$

$$< O(m(n+1)^{r-1}m^{r-1}(n+1)^{r-1}) = O(m^r(n+1)^{2(r-1)}).$$

In order to use for later analysis, let us now derive the exact computing time of Horner's method for two and three variables. Using equation (3.1) from the proof of Theorem 3.1 we find that the exact computing time for Horner's method with r=2 is

$$(n+1) \sum_{1\le i\le m} m + (i-1)n + 1$$

$$= (n+1)\left[(m+1)m + n \sum_{1\le i\le m} (i-1)\right]$$

$$= (n+1)\left[(m+1)m + \frac{n\cdot m(m-1)}{2}\right]$$

$$= \frac{m(n+1)}{2}(2(m+1) + n(m-1)) \qquad (3.2)$$

Again using equation (3.1) we find that the time for three variables becomes

$$(n+1)^2 \sum_{1\le i\le m} (m^2+2(i-1)mn+2m+(i-1)^2n^2+2(i-1)n+1)$$

$$= (n+1)^2\left[(m^2+2m+1)m+ n\sum_{1\le i\le m} (2(i-1)m +(i-1)^2n +2(i-1))\right]$$

$$= (n+1)^2\left[(m^2+2m+1)m+m^2n(m-1)+\frac{n^2m(m-1)(2m-1)}{6}+m(m-1)n\right]$$

$$= m(n+1)^2\left[(m+1)^2+nm(m-1)+\frac{n^2(m-1)(2m-1)}{6}+(m-1)n\right]$$

$$= m(n+1)^2\left[(m+1)^2+(nm+1)(m+1)+\frac{n^2(m-1)(2m-1)}{6}\right] . \qquad (3.3)$$

Now let us look at a new method for accomplishing the substitution. In this case evaluation points are chosen and P and Q are reduced until only a single variable remains in Q. Evaluations are then done using the conventional Horner's rule and the results are then interpolated to produce the correct polynomial. This technique has been successfully applied for speeding up other algebraic operations, e.g. see W. S. Brown [ 2 ], G. E. Collins [ 3 ] and E. Horowitz [ 7 ].

Algorithm SUBST(P,Q)

Input: $P(x_1,\ldots,x_{r-1})$, $Q(x_1,\ldots,x_r)$ $r \ge 1$;

```
Output:  $R(x_1,\ldots,x_{r-1}) = Q(x_1,\ldots,x_{r-1},P)$;

1) [Initial Case]     If $r = 1$ then use Horner's rule to
                      compute $Q(P)$ and return.
2) [Initialize]       $b \leftarrow -1$; $C \leftarrow 0$; $D \leftarrow 1$; $n_{r-1} \leftarrow \deg(P)$ in
                      $x_{r-1}$; $m_r \leftarrow \deg(Q)$ in $x_r$; $m_{r-1} \leftarrow \deg(Q)$ in
                      $x_{r-1}$;
3) [Choose a          $b \leftarrow b + 1$;
    new point]
4) [Evaluate]         $P^*(x_1,\ldots,x_{r-2}) \leftarrow P(x_1,\ldots,x_{r-2},b)$;
                      $Q^*(x_1,\ldots,x_{r-2},x_r) \leftarrow Q(x_1,\ldots,x_{r-2},b,x_r)$;
5) [Recursion]        $R^*(x_1,\ldots,x_{r-2}) \leftarrow \mathrm{SUBST}(P^*,Q^*)$;
6) [Interpolate]      $C(x_1,\ldots,x_{r-1}) \leftarrow \left\{\frac{R^* - C(x_1,\ldots,x_{r-2},b)}{D(b)}\right\}\cdot D(x_{r-1}) + C$;
7) [Update]           $D \leftarrow (x_{r-1}-b)\cdot D$; if $\deg(D) \leq m_r n_{r-1}+m_{r-1}$
                          then go to (3);
8) [End]              $R(x_1,\ldots,x_{r-1}) \leftarrow C$ and End.
```

<u>Theorem 3.2</u> Let $P(x_1,\ldots,x_{r-1})$ and $Q(x_1,\ldots,x_r)$ be dense polynomials in $r \geq 1$ variables with coefficients in GF(p). Further suppose that $n = \deg(P)$ in $x_i$, $1 \leq i \leq r-1$ and $m = \deg(Q)$ in $x_i$, $1 \leq i \leq r$. Then the computing time for SUBST(P,Q) is $O((m+1)^r(n+1)^r)$.

<u>Proof</u> We get the following table of computing times:

| Step | Time | Comments |
|---|---|---|
| (1) | m | since $P \in GF(p)$ |
| (2) | m | assuming any reasonable data structure for polynomials. |

Now steps (3) - (7) are executed mn + m times.

| | | |
|---|---|---|
| (3) | $m(n+1)$ | |
| (4) | $m(n+1)\{m+n+(n+1)^{r-1}+(m+1)^{r-1}\}$ | computing $b^i$ m and n times and then substituting. |
| (5) | $m(n+1)T(r-1)$ | |
| (6) | $\sum_{1\le i\le m(n+1)} i(m(n+1)+1)^{r-2}$ | $C(x_1,\ldots,x_{r-2},b)$ |
| | $+\sum_{1\le i\le m(n+1)} i$ | $D(x_{r-1})/D(b)$ |
| | $+\sum_{1\le i\le m(n+1)} i(m(n+1)+1)^{r-2}$ | $(R^* - C)\frac{D(x_{r-1})}{D(b)}$ |
| (7) | $\sum_{1\le i\le m(n+1)} i$ | $D(x_{r-1})(x_{r-1}-b)$ |

Thus

$$T(r) = 3m + mn + m(n+1)(m+n) + m(n+1)^r + m(n+1)T(r-1)$$
$$+ m(n+1)(m+1)^{r-1} + \sum_{1\le i\le m(n+1)} (2i + 2i(m(n+1) + 1)^{r-2})$$
$$= 3m + mn + m(n+1)\{(m+n) + (n+1)^{r-1} + (m+1)^{r-1}$$
$$+ T(r-1)\} + 2\,\frac{m(n+1)(m(n+1) + 1)}{2}$$

$$+ \frac{2m(n+1)(m(n+1)+1)}{2} (m(n+1)+1)^{r-2}$$

$$= 3m + mn + m(n+1)\{(m+n) + (n+1)^{r-1} + (m+1)^{r-1} + T(r-1)\}$$
$$+ m(n+1)(m(n+1)+1)(1 + (m(n+1)+1)^{r-2})$$

Let

$$a = m(n+1) + 1$$

and

$$b = 3m + mn + m(n+1)(m+n).$$

Then, working out the recurrence relation

$$T(r) = b + m(n+1)\{a(a^{r-2}+1) + (n+1)^{r-1} + (m+1)^{r-1} + T(r-1)\}$$

we get $T(r) =$

$$\begin{aligned} & b(1 + (a-1) + (a-1)^2 + \dots + (a-1)^{r-2}) \\ & + a((a-1)(a^{r-2}+1) + (a-1)^2(a^{r-3}+1) + \dots + (a-1)^{r-1}) \\ & + (n+1)^r(m + m^2 + \dots + m^{r-1}) \\ & + (a-1)(m+1)^{r-1} + (a-1)^2(m+1)^{r-2} + \dots + (a-1)^{r-1}(m+1) \\ & + (a-1)^{r-1}T(1) \end{aligned} \tag{3.4}$$

which we can approximate by

$$\begin{aligned} T(r) \le\ & b\left\{\frac{(a-1)^{r-1} - 1}{a - 2}\right\} + (r-1)a(a+1)^{r-1} + m^r(n+1)^{r-1} \\ & + (n+1)^r\left\{\frac{m^r - 1}{m - 1} - 1\right\} + (m+1)^r\left\{\frac{(n+1)^r - 1}{n} - 1\right\} \end{aligned} \tag{3.5}$$

I have retained this unwieldy expression in order to keep track of the constant. But asymptotically it follows from equation (3.5) and the values for a and b that the asymptotic time for SUBST is $O((m+1)^r(n+1)^r)$.

Now we derive the exact computing time for this method for two and three variables using equation (3.4).

$$\begin{aligned} T(2) &= b + 2a(a-1) + (n+1)^2 m + (a-1)(m+1) + (a-1)m \\ &= 3m + mn + m(n+1)(m+n) + 2m(n+1)(m(n+1)+1) \\ &\quad + m(n+1)^2 + m(m+1)(n+1) + m^2(n+1) \\ &= m(n+1)\{m + n + 2m(n+1) + 2 + n + 1 + n + 1 + m\} \\ &\quad + 3m + mn \\ &= m(n+1)\{2m(n+1) + 2m + 3n + 7\} \qquad (3.6) \end{aligned}$$

and

$$\begin{aligned} T(3) &= b(1+a-1) + a((a-1)(a+1) + (a-1)^2 2) \\ &\quad + (n+1)^3(m+m^2) + (a-1)(m+1)^2 + (a-1)^2(m+1) \\ &\quad + (a-1)^2 m \\ &= ab + a(a-1)(a+1) + 2a(a-1)^2 + (n+1)^3 m(m+1) \\ &\quad + (a-1)(m+1)^2 + (a-1)^2(m+1) + (a-1)^3 m \\ &= (a-1)^2\{2a + 2m + 1\} + (a-1)\{(m+1)^2 + a(a+1)\} \\ &\quad + ab + (n+1)^3 m(m+1) \qquad (3.7) \end{aligned}$$

Now let us analyze a third method which is perhaps the most obvious algorithm to use.

Algorithm STRAIGHT(P,Q)

```
     Input:  $P(x_1,\dots,x_{r-1})$, $Q(x_1,\dots,x_r)$
             $= \sum_{0\le i\le m_r} a_i(x_1,\dots,x_{r-1}) \cdot x_r^i$
     Output: $R(x_1,\dots,x_{r-1}) = Q(x_1,\dots,x_{r-1},P)$.

1) [Powers]      Compute $P^2, P^3, \dots, P^{m_r}$;  $R \leftarrow 0$;
2) [Main loop]   For $i \leftarrow 0, \dots, m_r$ do
                 $R \leftarrow R + a_i P^i$;
```

Theorem 3.3 Let $P(x_1,\dots,x_{r-1})$ and $Q(x_1,\dots,x_r)$ be dense polynomials in $r \ge 1$ variables with single precision coefficients. Further suppose $n = \deg(P)$ in $x_i$, $1 \le i \le r-1$ and $m = \deg(Q)$ in $x_i$, $1 \le i \le r$. Then the computing time for STRAIGHT(P,Q) is $O(m^r(n+1)^{r-1}\{(n+1)^{r-1} + (m+1)^{r-1}\})$.

Proof By [8], p. 19, with q replaced by (n+1), n replaced by m and r replaced by r-1, the time for step (1) is $O((n+1)^{2(r-1)}m^r)$. This can also be obtained directly from the formula

$$\sum_{1\le i\le m-1} (n+1)^{r-1}(in+1)^{r-1} < (n+1)^{r-1}(mn+1)^{r-1} \cdot m$$

$$< (n+1)^{r-1}(mn+m)^{r-1}m = O((n+1)^{2(r-1)}m^r). \qquad (3.8)$$

Now $P^i$ has $(in+1)^{r-1}$ terms while the $a_i$ have $(m+1)^{r-1}$ terms. Thus the total time for step (2) is given by

$$\sum_{1\le i\le m} (in+1)^{r-1}(m+1)^{r-1} < m(m+1)^{r-1}(mn+m)^{r-1}$$
$$= m^r(m+1)^{r-1}(n+1)^{r-1} \qquad (3.9)$$

Thus the total time for algorithm STRAIGHT is

$$m^r(n+1)^{2(r-1)} + m^r(m+1)^{r-1}(n+1)^{r-1}$$
$$= O(m^r(n+1)^{r-1}\{(n+1)^{r-1} + (m+1)^{r-1}\}).$$

Using equations (3.8) and (3.9) the exact time for algorithm STRAIGHT is

$$\sum_{1\le i\le m-1} (n+1)^{r-1}(in+1)^{r-1} + \sum_{1\le i\le m} (m+1)^{r-1}(in+1)^{r-1}. \qquad (3.10)$$

Thus for r=2 this exact time reduces to

$$(n+1)\left(\frac{nm(m-1)}{2} + m-1\right) + \left((m+1)\ \frac{nm(m+1)}{2} + m\right)$$
$$= \frac{(n+1)(m-1)(mn+2) + (m+1)(mn(m+1) + 2m)}{2}. \qquad (3.11)$$

For r=3 we get from equation (3.10)

$$(n+1)^2 \sum_{1\leq i\leq m-1} (in+1)^2 + (m+1)^2 \sum_{1\leq i\leq m} (in+1)^2$$

$$= [(n+1)^2 + (m+1)^2] \sum_{1\leq i\leq m} \{(in+1)^2\} - (mn+1)^2(n+1)^2$$

$$= [(n+1)^2 + (m+1)^2]\left[\frac{n^2m(m+1)(2m+1)}{6} + \frac{2nm(m+1)}{2} + m\right]$$

$$- (mn+1)^2(n+1)^2$$

$$= [(n+1)^2 + (m+1)^2]\left[nm(m+1)\left(\frac{n(2m+1)}{6} + 1\right) + m\right]$$

$$- (mn+1)^2(n+1)^2 \qquad (3.12)$$

Upon examing the exact computing times for HORNER versus STRAIGHT we see that for all m, n and $r \geq 1$ Horner's method requires fewer operations. Thus let us carefully compare the times for SUBST and HORNER.

Asymptotically we find that

$$\frac{\text{SUBST}}{\text{HORNER}} = \frac{(m+1)^r(n+1)^r}{m^r(n+1)^{2(r-1)}} = \frac{(m+1)^r}{m^r(n+1)^{r-2}} .$$

Thus we see that for $r > 2$, SUBST will outstrip HORNER's method as long as n is sufficiently large. Therefore, let us determine some representative values of n for this cut-off point.

Using equations (3.6) and (3.2) when r=2 we are comparing the time for SUBST which is

$$m(n+1)\{2m(n+1) + 2m + 3n + 7\}$$

and the time for HORNER which is

$$\frac{m(n+1)}{2}\{2(m+1) + n(m-1)\}$$

Some elementary algebra quickly shows that for all values of m, n $\geq$ 1 SUBST requires more multiplications than HORNER thereby supporting our previous conclusion.

Examing the case for r=3 equations (3.7) and (3.3) yield

$$(a-1)^2\{2a + 2m + 1\} + (a-1)\{(m+1)^2 + a(a+1)\} + ab$$
$$+ (n+1)^3 m(m+1)$$

for SUBST and

$$m(n+1)^2\{(m+1)^2 + (mn+1)(m-1) + \frac{n^2(m-1)(2m-1)}{6}\}$$

for HORNER.

The leading term for SUBST is

$$3m^3(n+1)^3$$

and for HORNER it is

$$\frac{m^3n^2(n+1)^2}{3}$$

Thus, SUBST will be better roughly for n > 9. Notice that the relation is independent of m, the degrees of Q, but simply relies on n. For a larger number of variables SUBST will continue to get better than HORNER for even smaller values of n.

In conclusion then for dense polynomials we have that

a) $r=1$, Horner's rule is optimal;

b) $r=2$, Horner's rule is always better by a constant factor;

c) for $r \geq 3$, SUBST is asymptotically the better method and for $r=3$ a value of approximately $n=9$ is the cut-off point.

## 4. Completely Sparse

We have seen in the previous section that an evaluation-interpolation algorithm can work quite efficiently when applied to dense polynomials. The question naturally arises as to whether or not this algorithm will work as well on polynomials which are less than dense. The difficulty is in precisely defining the class of polynomials we are referring to. A sparse polynomial has very few non-zero terms but this definition is really not quantitative enough to be satisfying (or what is more important not capable of being analyzed). However, even without a precise definition of sparsity we can say that SUBST will <u>not</u> work very well. The reason for this is that steps (3) - (7), the main loop, are governed by the degrees of the inputs.

The number of evaluation points which must be chosen is fixed despite the fact that the inputs or the result may be quite sparse. Thus, a method such as Horner's rule becomes more attractive for cases other than completely dense.

One quantitative definition of a sparse polynomial has been given by M. Gentleman in [5]. If P has t terms then he assumes that $P^n$ will have $\binom{t+n-1}{t-1}$ terms. For example one such polynomial P which obeys this definition is

$$P(x_1,\ldots,x_n) = x_1 + x_2 + \ldots + x_n$$

which I believe we would all agree is sparse, since it only has n out of $2^n$ possible terms. This definition is certainly an extreme case of polynomial growth, yet it is interesting to re-analyze our algorithms for this class of polynomials.

<u>Theorem 4.1</u> Let $P(x_1,\ldots,x_{r-1})$ be a completely sparse polynomial and $Q(x_1,\ldots,x_r) = \sum_{0\le i\le m} a_i(x_1,\ldots,x_{r-1}) \cdot x_r^i$ be a polynomial where each $a_i$ has s terms. Then the time needed to compute $Q(x_1,\ldots,x_{r-1},P)$ using algorithm HORNER is $O(st\binom{t+m}{t})$.

<u>Proof</u> Looking at the main loop in Horner's method, we need

to determine the number of terms of R. Initially R has s terms by the hypothesis. Then as i varies from m-1 to 0, we find that for

$i = m-1$, R has s terms

$i = m-2$, R has $s\binom{t}{t-1}$

$i = m-3$, R has $s\binom{t+1}{t-1}$

and it follows by our assumption of complete sparsity that $R^{(i)}$ has $s\binom{t+m-i-2}{t-1}$ terms where $m \geq 1$ for $0 \leq i \leq m-1$. Thus, the computing time for step (2) becomes

$$\sum_{0\leq i\leq m-1} s\binom{t+m-i-2}{t-1} t = st \sum_{0\leq i\leq m-1} \binom{t+i-1}{t-1} = st\binom{t+m}{t} .$$

Now let us compare this computing time with the result we get by using the straightforward method or Algorithm STRAIGHT.

Theorem 4.2 Let $P(x_1,\ldots,x_{r-1})$ be a completely sparse polynomial and $Q(x_1,\ldots,x_r) = \sum_{0\leq i\leq m} a_i(x_1,\ldots,x_{r-1}) \cdot x_r^i$ be a polynomial where each $a_i$ has s terms. Then the time needed to compute $Q(x_1,\ldots,x_{r-1},P)$ using algorithm STRAIGHT is $O((s+t)\binom{t+m}{t})$.

Proof In order to compute the powers of P, $P^2,\ldots,P^n$

the time needed is

$$\sum_{0\le i\le m-1} t\binom{t+i-1}{t-1} = t\left\{\binom{t+m}{t} - 1\right\}.$$

In order to compute the products $a_i P^i$ for $0 \le i \le m$ we need to do

$$\sum_{0\le i\le m} s\binom{t+i-1}{t-1} = s\binom{t+m}{t} \text{ multiplications}$$

and the number of additions required is $s\binom{t+m}{t}$. So the total time is $O((s+t)\binom{t+m}{t})$.

Now comparing these two methods their ratio is

$$\frac{\text{STRAIGHT}}{\text{HORNER}} = \frac{s+t}{st}$$

which reveals that algorithm STRAIGHT can actually be more efficient for these especially sparse classes of polynomials. Intuitively this springs from the fact that computing the powers takes all of the work. Thus Horner's method which continually adds more terms causes more arithmetic operations to be needed as the powers are computed. Unfortunately we cannot tell for most polynomials in a reasonable amount of time whether they satisfy Gentleman's definition of sparsity. Hence we cannot construct a simple decision procedure which wisely chooses between SUBST, HORNER or STRAIGHT.

Another approach for including sparse polynomials in the analysis is to assume that $P(x_1,\ldots,x_{r-1})$ has $t^{r-1}$ terms where t is a parameter which varies as $1 \leq t \leq n+1$. Then the computing time for algorithm HORNER becomes $m^r t^{2(r-1)}$ and we can then produce a rule which tells us when Horner's method will perform better than SUBST. Namely

$$\frac{\text{HORNER}}{\text{SUBST}} = \frac{m^r t^{2(r-1)}}{[(m+1)(n+1)]^{r+1}}$$

and HORNER is better than SUBST when the number of non-zero terms in P, namely $t^{r-1}$, satisfies

$$t^{r-1} < \sqrt{m(n+1)^{r+1}} \, . \tag{4.1}$$

This equation is quite useful since it provides a test which can be computed in linear time and which adequately discriminates between the two methods, SUBST versus HORNER.

Another extreme form of a polynomial substitution problem would have Q in the form

$$Q(x_1,\ldots,x_r) = x_r^m + \bar{Q}(x_1,\ldots,x_{r-1}) \, .$$

Computing $Q(x_1,\ldots,x_{r-1},P)$ in this case really reduces to computing polynomial powers efficiently, namely $P(x_1,\ldots,x_{r-1})^m$. For a comparison of algorithms for polynomial powering which includes sparse and dense polynomials as subcases, see Horowitz and Sahni [8].

In conclusion, Horner's method still remains the most versatile of the substitution algorithms. The test given in equation (4.1) allows us to modify SUBST so that at each level of recursion we can decide whether to continue evaluating or to switch to Horner's method. Thus within our existing symbol manipulation system both SUBST and Horner's method can be made available with an automatic test for deciding when to use either algorithm.

## References

1. Borodin, Allan "Horner's Rule is Uniquely Optimal", Theory of Machines and Computations, ed. by Z. Kohavi and A. Paz, Academic Press, 1971, pp. 45-59.

2. Brown, W.S. "On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors", J.ACM, Vol. 18, No. 4, October 1971, pp. 478-504.

3. Collins, G.E. "The Calculation of Multivariate Polynomial Resultants", J.ACM, Vol. 18, No. 4, October 1971, pp. 515-532.

4. Collins, G.E. "The SAC-1 Polynomial System", University of Wisconsin, Comp. Sci. Tech. Report No. 115, March 1971.

5. Gentleman, W.M. "Optimal Multiplication Chains for Computing a Power of a Symbolic Polynomial", Mathematics of Computation, Vol. 26, No. 120, October 1972, pp. 935-940.

6. Hall, A.D. "The ALTRAN System for Rational Function Manipulation - A Survey", Proc. Second Symposium on Symbolic and Algebraic Manipulation, ACM, N.Y. 1971, pp. 153-157.

7. Horowitz, E. "The Efficient Calculation of Polynomial Powers", J.CSS (to appear).

8. Horowitz, E. and Sahni, S. "On the Computation of Powers of a Class of Polynomials", C.S. Tech. Report No. 72-143, Cornell University, Ithaca, August 1972.

9. Petrick, S., ed. Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, ACM, New York, 1971.