Labeled Trees and the Efficient Computation of Derivations

Robert Grossman* and Richard G. Larson†
University of Illinois at Chicago

May 16, 2004

This is a draft of a paper which later appeared in Proceedings of 1989 International Symposium on Symbolic and Algebraic Computation, ACM, 1989, pp. 74-80.

1 Introduction

This paper is concerned with the effective parallel symbolic computation of operators under composition. Examples include differential operators under composition and vector fields under the Lie bracket. In general, such operators do not commute. An important problem is to find efficient algorithms to write expressions involving noncommuting operators in terms of operators which do commute. If the original expression enjoys a certain symmetry, then naive rewriting requires the computation of terms which in the end cancel. In [8], we gave an algorithm which in some cases is exponentially faster than the naive expansion of the noncommutating operators. The purpose of this paper is show how that algorithm can be naturally parallelized.

In Section 2, we give a careful statement of the problem. In Section 3, we discuss data structures consisting of formal linear combinations of rooted labeled trees. We define a multiplication on rooted labeled trees, thereby making the set of these data structures into an associative algebra. We then define an algebra homomorphism from the original algebra of operators into this algebra of trees. In Section 4, we describe an algebra homomorphism

*Supported in part by NASA Grant NAG 2–513.
†Supported in part by NSF Grant DMS 870–1085

from the algebra of trees into the algebra of differential operators. The cancellation which occurs when noncommuting operators are expressed in terms of commuting ones occurs naturally when the operators are represented using this data structure. This leads to an algorithm which, for operators which are derivations, speeds up the computation exponentially in the degree of the operator. This is described in Section 5. Sections 3–5 follow the treatment of [8]. In Section 6, we show how the algebra of trees leads naturally to a parallel version of the algorithm.

Here is a concrete example of the type of computations we are concerned with. Fix three vector fields E_1, E_2, E_3 in $\mathbf{R}^N$ with polynomial coefficients a_i^j:

$$E_i = \sum_{j=1}^{N} a_i^j \frac{\partial}{\partial x_j}, \qquad \text{for } i = 1,\ 2,\ 3.$$

Considering the vector fields as first-order differential operators, it is natural to form higher-order differential operators from them, such as the third-order differential operator

$$p = E_3E_2E_1 - E_3E_1E_2 - E_2E_1E_3 + E_1E_2E_3.$$

Writing this differential operator in terms of the $\partial/\partial x_1, \ldots, \partial/\partial x_N$ yields a first-order differential operator because of the symmetry of the expression p causes all second- and third-order terms to cancel.

In this paper we analyse an algorithm for expressing differential operators p in terms of the commuting derivations $\partial/\partial x_1, \ldots, \partial/\partial x_N$ in such a way that second and third order terms which cancel are not computed. In the example above, the naive computation would require the computation of $24N^3$ terms, while the algorithm we describe here would involve just the computation of the $6N^3$ terms which do not cancel.

We conclude this introduction with some remarks.

1. In actual applications expressions possessing symmetry arise more often than not. For example, Lie brackets of vector fields possess a great deal of symmetry. The algorithm we discuss is designed to take advantage of such symmetries, if they are present, without the necessity of explicitly identifying the symmetries.

2. Once a set of data structures has been given an algebraic structure, it becomes natural to view algorithms concerned with simplification as simply the factoring of a map into the composition of a map into the algebra of these data structures, and a map from this algebra. This is

the simple idea which is at the basis of the algorithm we describe. We expect that this idea will find application elsewhere.

3. See [4] and [3] for previous work on the simplification of expressions. See [9] and the references contained there for previous work on parallel symbolic computation.

2 Higher-order derivations

In this section we give a careful statement of the problem and state the main result. Let R be a commutative algebra with unit over the field k. (Throughout this paper k is a field of characteristic 0.) A *derivation* of the algebra R is a linear map D of R to itself satisfying

$$D(ab) = aD(b) + bD(a), \qquad \text{for all } a, b \in R.$$

Let $D_1, \ldots, D_N$ be N commuting derivations of R, that is, for $i, j = 1, \ldots, N$,

$$D_i D_j a = D_j D_i a, \qquad \text{for all } a \in R.$$

Suppose that we are also given M derivations $E_1, \ldots, E_M$ of R which can be expressed as R-linear combinations of the derivations D_i; that is, for $j = 1, \ldots, M$,

$$E_j = \sum_{\mu=1}^{N} a_j^\mu D_\mu, \qquad \text{where } a_j^\mu \in R. \tag{1}$$

We are interested in writing higher-order derivations generated by the $E_1, \ldots, E_M$ in terms of the commuting derivations $D_1, \ldots, D_N$. More formally, let $k\!<\!E_1, \ldots, E_M\!>$ denote the free associative algebra in the symbols $E_1, \ldots, E_M$ and let $\mathbf{Diff}(D_1, \ldots, D_N; R)$ denote the space of formal linear differential operators with coefficients from R; that is, $\mathbf{Diff}(D_1, \ldots, D_N; R)$ consists of all finite formal expressions

$$\sum_{\mu_1=1}^{N} a_{\mu_1} D_{\mu_1} + \sum_{\mu_1,\mu_2=1}^{N} a_{\mu_1\mu_2} D_{\mu_2} D_{\mu_1} + \cdots$$

where $a_{\mu_1}, a_{\mu_1\mu_2}, \ldots \in R$. We let

$$\chi : k\!<\!E_1, \ldots, E_M\!> \longrightarrow \mathbf{Diff}(D_1, \ldots, D_N; R)$$

denote the map which sends $p \in k\!<\!E_1, \ldots, E_M\!>$ to the linear differential operator $\chi(p)$ obtained by performing the substitution (1) and simplifying using the fact that the D_μ are derivations of R.

Suppose $p \in k<E_1, \ldots, E_M>$ is of the form

$$p = \sum_{i=1}^{l} p_i,$$

where each term p_i is of degree m. The naive computation of $\chi(p)$ would compute $\chi(p_i)$, for $i = 1, \ldots, l$. This would yield $l\, m!\, N^m$ terms. Assume $\mathrm{Cost_A}(p)$, the cost of applying algorithm A to simplify $p \in k<E_1, \ldots, E_M>$, is proportional to the number of differentiations and multiplications. Then

$$\mathrm{Cost_{NAIVE}}(p) = O(lm\, m!\, N^m)$$

In Section 5, we describe an algorithm which preprocesses an expression p in such a way that any terms which cancel after the substitution (1) are not computed. We show:

Theorem 1 *Assume that*

(i) *p is the sum of $l = 2^{m-1}$ terms, each homogenous of degree m;*

(ii) *$L = \chi(p)$ is a linear differential operator of degree 1.*

(iii) *$m, N \to \infty$ in such a way that $2^{m-2}m \ll N^m$.*

Then

$$\frac{\mathrm{Cost_{BETTER}}(p)}{\mathrm{Cost_{NAIVE}}(p)} = O\left(\frac{1}{m2^{m-1}}\right).$$

In Section 6, we describe how this precomputation can be done as a parallel computation.

Observe that a Lie bracket of degree m on $\mathbf{R}^N$, for large enough N, satisfies the hypotheses of the theorem.

3 Trees and differential operators

In this section we describe the connection between algebras and trees which is essential for the description of the data structures which we use in the next section, and for the analysis of the algorithms which use those data structures.

By a tree we mean a rooted finite tree [10]. If $\{E_1, \ldots, E_M\}$ is a set of symbols, we will say a tree is *labeled with* $\{E_1, \ldots, E_M\}$ if every node of the tree other than the root has an element of $\{E_1, \ldots, E_M\}$ assigned to it. We denote the set of all trees labeled with $\{E_1, \ldots, E_M\}$ by $\mathcal{LT}(E_1, \ldots, E_M)$.

Let $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ denote the vector space over k with basis $\mathcal{LT}(E_1, \ldots, E_M)$. We show that this vector space is a graded connected algebra.

We define the multiplication in $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ as follows. Since the set of labeled trees form a basis for $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$, it is sufficient to describe the product of two labeled trees. Suppose t_1 and t_2 are two labeled trees. Let $s_1, \ldots, s_r$ be the children of the root of t_1. If t_2 has $n+1$ nodes (counting the root), there are $(n+1)^r$ ways to attach the r subtrees of t_1 which have $s_1, \ldots, s_r$ as roots to the labeled tree t_2 by making each s_i the child of some node of t_2, keeping the original labels. The product $t_1 t_2$ is defined to be the sum of these $(n+1)^r$ labeled trees. It can be shown that this product is associative, and that the tree consisting only of the root is a multiplicative identity; see [5].

We can define a grading on $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ by letting $k\{\mathcal{LT}_n(E_1, \ldots, E_M)\}$ be the subspace of $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ spanned by the trees with $n+1$ nodes. The following theorem is proved in [6].

Theorem 2 *$k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ is a graded connected algebra.*

If $\{E_1, \ldots, E_M\}$ is a set of symbols, then the free associative algebra $k<E_1, \ldots, E_M>$ is a graded connected algebra, and there is an algebra homomorphism

$$\phi : k<E_1, \ldots, E_M> \to k\{\mathcal{LT}(E_1, \ldots, E_M)\}.$$

The map ϕ sends E_i to the labeled tree with two nodes: the root, and a child of the root labeled with E_i; it is then extended to all of $k<E_1, \ldots, E_M>$ by using the fact that it is an algebra homomorphism.

We say that a rooted finite tree is heap-ordered in case there is a total ordering on all nodes in the tree such that each node procedes all of its children in the ordering. We say such a tree is labeled with $\{E_1, \ldots, E_M\}$ in case every element, except the root, has an element of $\{E_1, \ldots, E_M\}$ assigned to it. Let $k\{\mathcal{LHOT}(E_1, \ldots, E_M)\}$ denote the vector space over k whose basis consists of labeled heap-ordered trees. It turns out that $k\{\mathcal{LHOT}(E_1, \ldots, E_M)\}$ is also a graded connected algebra using the same multiplication defined above. See [6] for a proof of the following theorem.

Theorem 3 *The map*

$$\phi : k<E_1, \ldots, E_M> \to k\{\mathcal{LHOT}(E_1, \ldots, E_M)\}$$

is injective.

4 Simplification of higher order derivations

In this section we define a map

$$\psi : k\{\mathcal{LT}(E_1, \ldots, E_M)\} \to \mathbf{Diff}(D_1, \ldots, D_N; R).$$

We do this in several steps.

Step 1. Given a labeled tree $t \in \mathcal{LT}_m(E_1, \ldots, E_M)$, assign the root the number 0 and assign the remaining nodes the numbers $1, \ldots, m$. From now on we identify the node with the number assigned to it. Let $k \in$ nodes t , and suppose that $l, \ldots, l'$ are the children of k. Fix $\mu_l, \ldots, \mu_{l'}$ with

$$1 \leq \mu_l, \ldots, \mu_{l'} \leq N$$

and define

$$\begin{aligned} R_t(k; \mu_l, \ldots, \mu_{l'}) &= D_{\mu_l} \cdots D_{\mu_{l'}} a_{\gamma_k}^{\mu_k} \\ &\quad \text{if } k \text{ is not the root} \\ &= D_{\mu_l} \cdots D_{\mu_{l'}} \\ &\quad \text{if } k \text{ is the root .} \end{aligned}$$

We abbreviate this to $R_t(k)$ or $R(k)$. Observe that $R_t(k) \in R$ for $k > 0$.

Step 2. Define

$$\psi(t) = \sum_{\mu_1, \ \ldots, \ \mu_m = 1}^{N} R(m) \cdots R(1) R(0).$$

Step 3. Extend ψ to all $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ by K-linearity.

The next three propositions describe fundamental properties of the map ψ. Note that the next proposition is an example of simplification by factoring χ through the set of labeled trees: we will see that often ψ and ϕ together are cheaper to compute than χ.

Proposition 4 *(i) The map ψ is an algebra homomorphism.*

(ii)

$$\chi = \psi \circ \phi.$$

Proof: The proof of (i) is a straightforward verification and is contained in [7]. Since χ and $\psi \circ \phi$ agree on the generating set $E_1, \ldots, E_M$, part (ii) follows from part (i).

5 The cost of computing derivations

In this section, we briefly review the discussion in [8] on the work required to write an expression composed of noncommuting operators in terms of commuting operators. This will prepare us for the next section in which we consider the cost to simplify such expression given several processors. We make the following asssumptions: $p \in k<E_1, \ldots, E_M>$ is of the form

$$p = \sum_{i=1}^{l} p_i,$$

where each term p_i is of degree m; the cost of a multiplication is one unit and the cost of a differention is one unit; the cost of an addition is zero units; and the cost of adding a node to a tree is one unit, so that the cost of building a tree $t \in \mathcal{LT}_m(E_1, \ldots, E_M)$ is m units.

Proposition 5 *(i) $\chi(p)$ contains $lm!\, N^m$ terms.*

(ii) The cost of computing $\chi(p)$ is $2lm\, m!\, N^m$.

PROOF: Suppose p_i is of the form $E_{\gamma_m} \cdots E_{\gamma_1}$, for some indices $1 \leq \gamma_1, \ldots, \gamma_m \leq M$. Then $\chi(p_i)$ is equal to

$$(\sum_{\mu_m=1}^{N} a_{\gamma_m}^{\mu_m} D_{\mu_m}) \cdots (\sum_{\mu_1=1}^{N} a_{\gamma_1}^{\mu_1} D_{\mu_1}).$$

After expansion there are $m!N^m$ terms, each of which involves m differentions and m multiplications.

Proposition 6 *The cost of computing $\phi(p)$ is $lm\, m!$.*

PROOF: A monomial of degree m is sent to the sum of $m!$ labeled trees under the map ϕ. This follows easily by induction and is contained in [5]. By the assumptions above the cost of constructing a labeled tree with m nodes (in addition to the root) is m units. Therefore the total cost is $lm\, m!$.

Proposition 7 *Let $\sigma = \phi(p)$, and denote by $|\sigma|$ the number of labeled trees with non-zero coefficients in σ. Then the cost of computing $\psi(\sigma)$ is $2m|\sigma|N^m$.*

PROOF: Fix a labeled tree

$$t \in \mathcal{LT}_m(E_1, \ldots, E_M).$$

From the definition of the map ψ we see that the cost of computing $\psi(t)$ is $2mN^m$, and hence the total cost is $2m|\sigma|N^m$.

Combining these three propositions gives

Theorem 8 *Under the assumptions above, the cost* $\mathrm{Cost}_{\mathrm{NAIVE}}(p)$ *of computing*

$$\chi(p) = \sum_{i=1}^{l} \chi(p_i)$$

is $2lmm!\,N^m$*, while the cost* $\mathrm{Cost}_{\mathrm{BETTER}}(p)$ *of computing*

$$L = \psi \circ \phi(p)$$

is $lm\,m! + 2m|\sigma|N^m$.

Theorem 1 now follows.

6 Computing derivations with several processors

In the previous sections, we have shown how trees are naturally associated with the symbolic computation of higher order derivations. In this section, we show how trees also lead to natural parallel algorithms for symbolic computation. Rather than try to state and prove the sharpest results, we are content to state and prove an illustrative theorem of this type.

The problem is to rewrite the expression $p \in k{<}E_1, \ldots, E_M{>}$ in terms of commuting operators when several processors are available. As usual let $\chi(p) \in \mathbf{Diff}(D_1, \ldots, D_N; R)$ denote the resulting linear differential operator. Make the following asssumptions:

1. $p \in k{<}E_1, \ldots, E_M{>}$ is of the form

   $$p = \sum_{i=1}^{l} p_i,$$

   where each term p_i is of degree m.

2. The cost of a multiplication or addition is one unit and the cost of a differentiation is one unit; the cost of adding a node to a tree is one unit, so that the cost of building a tree $t \in \mathcal{LT}_m(E_1, \ldots, E_M)$ is $m+1$ units.

3. We assume that $p \in k<E_1, \ldots, E_M>$ is in its simplest form; in other words, any term $E_{\gamma_m} \cdots E_{\gamma_1}$ appears at most once.

4. We assume that there is one processor available for each labeled tree which arises in the computation.

Notation. Each term p_i in $p \in k<E_1, \ldots, E_M>$ is of the form

$$c_i E_{\gamma_m} \cdots E_{\gamma_1}, \qquad c_i \in k.$$

`LabelIndex` is defined to be an index taking values between 1 and m. If `LabelIndex` $= j$, then we denote by `LabelIndex`(p_i) the label E_{γ_j} in the term p_i of p. In the precomputation, we assign one processor for each rooted labeled tree in $\mathcal{LT}(E_1, \ldots, E_M)$. Each processor u has the following data structures associated to it:

1. for each label $E_j \in \{E_1, \ldots, E_M\}$, a list of processors, denoted `ProcessorList`(E_j) or `ProcessorList`$(u)(E_j)$;

2. an array `TermCount` containing counters such that `TermCount`$(u)[i]$ gives the number of times that term p_i in the polynomial $p \in k<E_1, \ldots, E_M>$, has contributed to the tree u;

3. a variable `TreeCoefficient`(u), which will be used to store the coefficient k of the tree t in $\sigma = \phi(p)$.

We say that the processor $u = u_t$ is *active* in case $\sum_{i=1}^{l}$ `TermCount`$(u)[i] > 0$. In other words, a processor $u = u_t$, where $t \in \mathcal{LT}_k(E_1, \ldots, E_M)$, is active in case its `TermCount` array has some positive entry.

We begin by describing a precomputation.

Step 1. We associate a processor $u = u_t$ to each tree in $LT_k(E_1, \ldots, E_M)$, for $k = 1, \ldots, m$.

Step 2. Let u_t be the processor assigned to the tree $t \in \mathcal{LT}_k(E_1, \ldots, E_M)$, for $k < m$, in Step 1, with labels $E_{\gamma_k}, \ldots, E_{\gamma_1}$. Let $E_{\gamma_{k+1}}$ be a label. The tree t yields $k+1$ trees labeled with $E_{\gamma_{k+1}}, \ldots, E_{\gamma_1}$ which arise by attaching the node labeled $E_{\gamma_{k+1}}$ to the tree t in all possible ways. Since these are labeled trees, they have already been assigned a processor by the step above. Let $u_1, \ldots, u_{k+1}$ denote these processors. In this step, we create the list `ProcessorList`$(E_{\gamma_{k+1}}, u)$ containing the processors $u_1, \ldots, u_{k+1}$. We do this for each label $E_{\gamma_{k+1}} \in \{E_1, \ldots, E_M\}$.

```
(* Step 0 *)
for each processor u do simultaneously
    for i := 1 to l do
        TermCount(u)[i] := 0;
    end;
end;

(* Step 1 *)
LabelIndex := 1;
for i := 1 to l do
    TermCount(u_i)[i] := 1;
end;
(* In Step 1, u_i denotes the tree with two nodes,
in which the node other than the root is
labeled with LabelIndex(p_i). *)

(* Step 2 *)
for LabelIndex := 1 to m - 1 do
    for each active processor u = u_t for which
        t has LabelIndex + 1 nodes do simultaneously
        for i := 1 to l do
            for all u' ∈ ProcessorList(LabelIndex(p_i), u) do
                TermCount(u')[i] := TermCount(u')[i]
                  +TermCount(u)[i];
            end;
        end;
    end;
end;

(* Step 3 *)
for each active processor u = u_t for which
    t has m + 1 nodes do simultaneously
    TreeCoefficient(u) := 0;
    for i := 1 to l do
        TreeCoefficient(u) := TreeCoefficient(u)
          +c_i * TermCount(u)[i];
    end;
end;
```

Figure 1: The Parallel Computation of ϕ.

We give the algorithm to do the parallel computation of ϕ in Figure 1. We make two remarks. First, write conflicts are possible in Step 2 of the algorithm. Indeed, consider the addition of $\texttt{TermCount}(u)[i]$ to $\texttt{TermCount}(u')[i]$ by processor u. Suppose that processor u' is associated with tree t'. Then the number of possible increments of $\texttt{TermCount}(u')[i]$, if u' is associated with a tree with $k+1$ nodes, is at most k. This is because one processor is associated with each tree that arises by deleting one leaf from t'. A processor associated with a tree with k nodes will access the element $\texttt{TermCount}(u)[i]$ of k other processors. Therefore a processor u will need to wait at most lm cycles to access the entry $\texttt{TermCount}(u')[i]$, and will need to access at most m such entries for each i.

Second, using Brent's algorithms for the parallel computation of arithmetic expressions [1], it is possible to compute $\psi(t)$ in parallel. Let $\sigma = \phi(p)$ and recall that the number of operations to compute $\psi(\sigma)$ is $O(m\,|\sigma|N^m)$ by Proposition 7. Therefore, given sufficiently many processors, $\psi(\sigma)$ can be computed in time $O(\log_2(m\,|\sigma|N^m))$.

Proposition 9 *The cost of computing $\phi(p)$ according to the algorithm in Figure 1 is $O(l^2m^3)$.*

Proof: Step 0 and Step 3 take time $O(l)$. Step 1 takes time $O(l^2)$. If $t \in \mathcal{LT}_k(E_1, \ldots, E_M)$ and $u = u_t$, then the following estimate holds for the inner loop of Step 2. The outer loop is repeated m times. The next sequential loop is repeated l times. Since the length of `ProcessorList` is at most m, the next sequential loop is repeated at most m times. By the first remark above, each of the at most m iterations of this loop will need to wait at most lm time units to execute. Therefore the total execution time for Step 2 is bounded by $O(l^2m^3)$. This completes the proof of the proposition.

Recall that by Proposition 6, $\phi(p)$ can be computed in serial time $O(lm\,m!)$. Comparing this to the cost of the algorithm above gives

Theorem 10

$$\frac{\text{Cost}_{\text{serial }\phi\text{-algorithm}}(p)}{\text{Cost}_{\text{parallel }\phi\text{-algorithm}}(p)} = O\left(\frac{lm^2}{m!}\right).$$

References

[1] R. P. Brent, *The parallel evaluation of general arithmetic expressions,* J. Assoc. Comp. Machinery **21** (1974), 201–206.

[2] B. Buchberger et. al. , "Computer algebra: symbolic and algebraic computation," Springer, Wien, 1983.

[3] B. Buchberger and R. Loos, *Algebraic Simplification,* in "Computer algebra: symbolic and algebraic computation," ed. B. Buchberger et. al., Springer, Wien, 1983, 11–43.

[4] B. F. Caviness, *On canonical forms and simplification,* J. Assoc. Computing Machinery **17** (1970), 385–396.

[5] R. Grossman, *Evaluation of expressions involving higher order derivations,* Center For Pure and Applied Mathematics, PAM - 367, University of California, Berkeley, submitted for publication.

[6] R. Grossman and R. G. Larson, *Hopf-algebraic structures of families of trees,* J. of Algebra, to appear.

[7] R. Grosssman and R. G. Larson, *Solving Nonlinear Equations From Higher Order Derivations in Linear Stages,* Advances in Math., to appear.

[8] R. Grossman and R. G. Larson, *Labeled Trees and the Algebra of Differential Operators,* to appear in "Algorithms and Graphs," AMS, 1989.

[9] C. G. Ponder, *Evaluation of "Performance Enhancements" in Algebraic Manipulation Systems,* University of California at Berkeley Computer Science Report UCB/CSD 88/438, 1988.

[10] R. E. Tarjan, "Data Structures and Network Algorithms," SIAM, Philadelphia, 1983.