Symbolic computation of derivations using labeled trees

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

October 10, 1989

This is a draft of a paper which later appeared in the Journal of Symbolic Computation, Volume 13, pp. 511-523, 1992.

1 Introduction

This paper discusses the effective symbolic computation of operators under composition. Examples include differential operators under composition and vector fields under the Lie bracket. Such operators in general do not commute, but are often rewritten in terms of other 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 this paper we analyse 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. 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.

*Supported in part by NASA Grant NAG 2–513 and by NSF Grant DMS-8904740.
†Supported in part by NSF Grant DMS 870–1085 and by NSF Grant DMS-8904740.

We first consider a concrete example. 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 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 requires the computation of $24N^3$ terms, while the algorithm we describe here involves the computation of just 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, as does the Laplacian

   $$L = E_1E_1 + E_2E_2 + E_3E_3$$

   built from the vector fields. 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 factoring a map through the algebra of these data structures. 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. The space of operators on a linear space is not only an algebra but also a coalgebra; that is, it is the dual of an algebra. The algebra of data structures mentioned above also has a coalgebra structure. Although this fact plays a relatively minor role in the simple algorithms discussed in this paper, it does play a crucial role for other algorithms we have studied.

Section 2 gives a careful statement of one of our main results. Section 3 reviews some background material on Lie algebras and Hopf algebras. Section 4 examines a natural Hopf algebraic structure on families of labeled trees and defines a homomorphism from the Hopf algebra of differential operators generated by vector fields to the Hopf algebra of labeled trees. Section 5 describes the simplification algorithm; its cost is computed in Section 6. Section 7 gives a parallel version of the simplification algorithm.

This work described in this paper was announced in [5] and [6].

2 Higher-order derivations

In this section we give a careful statement of the problem, and state one of the main results. 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 from 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, $\mathbf{Diff}(D_1, \ldots, D_N; R)$ is the algebra of all maps $R \to R$ generated by the maps D_μ and $L(a)$, $a \in R$. (The map $L(a)$, $a \in R$, is defined by $L(a)(b) = ab$ for $b \in R$.) We let

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

denote the map which sends $p \in k{<}E_1, \ldots, E_M{>}$ to the linear differential operator $\chi(p)$ obtained by performing the substitutions (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 $\mathrm{Cost}_{\mathrm{A}}(p)$, 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

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

In Section 5 we describe and in Section 6 we analyze an algorithm which preprocesses an expression p in such a way that any terms which cancel after the subtitutions (1) are not computed. We show

Theorem 2.1 *Assume*

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

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

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

Then

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

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

In Section 7, we show how this algorithm can be naturally parallelized.

3 Lie algebras and Hopf algebras

In the next section we will give a Hopf algebra structure on trees and differential operators. In this section we summarize the definitions and needed properties of Hopf algebras. Throughout the rest of this paper, k is a field of characteristic 0.

Definition 3.1 *A* Lie algebra *is a vector space L over the field k, together with a bilinear map $[-,-] : L \times L \to L$ satisfying*

$$\begin{aligned} [x,y] + [y,x] &= 0 \qquad && \text{for all } x, y \in L \\ [x,[y,z]] + [y,[z,x]] + [z,[x,y]] &= 0 \qquad && \text{for all } x, y, z \in L. \end{aligned}$$

The standard example of a Lie algebra is the following. Let A be an associative algebra over k, and let A^- be the Lie algebra with the same underlying vector space as A, with $[-,-]$ defined by $[x,y] = xy - yx$ for x, $y \in A$. It can be proved (see [7]) that every Lie algebra L is isomorphic to a sub-Lie algebra of A^- for some associative algebra A.

Every Lie algebra L is contained in an associative algebra $U(L)$, called the *universal enveloping algebra* of L, which is characterized by the fact that every Lie algebra homomorphism $L \to A^-$ induces a unique associative algebra homomorphism $U(L) \to A$. The algebra $U(L)$ is analogous to the free associative algebra $k<X>$ generated by a set X. In fact, one can construct the free Lie algebra $\mathcal{L}(X)$ generated by the set X, and prove that $k<X> = U(\mathcal{L}(X))$.

A basis for the associative algebra $U(L)$ is described in

Theorem 3.2 (Poincaré-Birkhoff-Witt) *Let L be a Lie algebra with ordered basis $\{x_1, \ldots, x_n, \ldots\}$. Then $\{ x_{i_1}^{e_1} \cdots x_{i_t}^{e_t} \mid t \geq 0, i_1 < \ldots < i_t, 0 < e_i \}$ is a basis for $U(L)$.*

See [7] for a proof.

If A is an associative algebra with identity over the field k, we can describe the structure of A with the linear maps $\mu : A \otimes A \to A$ defined by $\mu(a \otimes b) = ab$ for a, $b \in A$, and $\eta : k \to A$ defined by $\eta(x) = x1$ for $x \in k$. The fact that multiplication is associative can be restated as $\mu \circ (I \otimes \mu) = \mu \circ (\mu \otimes I)$, where $I : A \to A$ denotes the identity map. The fact that 1 is the multiplicative

identity can be restated as $\mu \circ (\eta \otimes I) = \mu \circ (I \otimes \eta) = I$, where we identify A with $k \otimes A$ and $A \otimes k$ via the canonical isomorphisms. The dual notion to an algebra is a coalgebra:

Definition 3.3 *A* coalgebra *over the field k is a vector space C over k, together with maps $\Delta : C \to C \otimes C$ and $\epsilon : C \to k$ such that $(\Delta \otimes I) \circ \Delta = (I \otimes \Delta) \circ \Delta$ and $(\epsilon \otimes I) \circ \Delta = (I \otimes \epsilon) \circ \Delta = I$, where $I : C \to C$ denotes the identity map, and we identify C with $k \otimes C$ and $C \otimes k$ via the canonical isomorphisms. The map Δ is called the* comultiplication *of C, and the map ϵ is called the* counit *of C.*

The coalgebra C is said to be cocommutative *if $\Delta = T \circ \Delta$ where $T : C \otimes C \to C \otimes C$ is defined by $T(a \otimes b) = b \otimes a$.*

If A is a finite dimensional algebra with multiplication μ and unit η, then the linear dual A^* is a coalgebra, with comultiplication μ^* and counit η^*. A Hopf algebra is both an algebra and a coalgebra, together with additional structure.

Definition 3.4 *A* Hopf algebra *is a vector space A over the field k, together with maps $\mu : A \otimes A \to A$, $\eta : k \to A$, $\Delta : A \to A \otimes A$, and $\epsilon : A \to k$, such that*

i) *μ and η define an (associative) algebra structure on A;*

ii) *Δ and ϵ define a coalgebra structure on A;*

iii) *the maps Δ and ϵ are algebra homomorphisms.*

If L is a Lie algebra, the universal enveloping algebra $U(L)$ is a Hopf algebra. Comultiplication is defined by $\Delta(x) = 1 \otimes x + x \otimes 1$ for $x \in L$, and extended to $U(L)$ by using the facts that L generates $U(L)$ as an algebra and that Δ is an algebra homomorphism. The counit is defined by $\epsilon(x) = 0$ for $x \in L$.

A vector space V over k is said to be *graded* if it is the direct sum of a family of subspaces indexed by the natural numbers:

$$V = \bigoplus_{n \geq 0} V_n.$$

A graded vector space V is said to be *connected* if $V_0 \cong k$. A graded vector space V is said to be *positively graded* if $V_0 = 0$. The tensor product of two

graded vector spaces is graded as follows:

$$(V \otimes W)_n = \sum_{p+q=n} V_p \otimes W_q.$$

An algebra (coalgebra, Hopf algebra, Lie algebra, ...) is graded if it is a graded vector space, and if all of the structure-defining maps preserve the grading. It can be easily seen that if L is a positively graded Lie algebra, then $U(L)$ is a graded connected Hopf algebra.

Definition 3.5 *Let A be a Hopf algebra.*

$$P(A) = \{\, a \in A \mid \Delta(a) = 1 \otimes a + a \otimes 1 \,\}.$$

If A is a Hopf algebra, it can be proved that $P(A)$ is a sub Lie algebra of A^-. If L is a Lie algebra, it can be proved that $L = P(U(L))$. A partial converse to this fact plays a key role in the structure of Hopf algebras:

Theorem 3.6 (Milnor-Moore) *Let A be a graded connected cocommutative Hopf algebra. Then $A \cong U(P(A))$ as Hopf algebras.*

See [8] or [9] for a proof of this.

4 Trees and Hopf algebras

In this section we describe the connection between Hopf algebras and trees which is essential for the description of the data structures which we introduce 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. 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 give this vector space the structure of a graded connected cocommutative Hopf algebra, and describe its primitive elements.

We define the multiplication in $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ as follows. Since the set of labeled trees forms 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 [2] or [3] for a proof.

We define the comultiplication on $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ as follows. Let t be a labeled tree, and let $s_1, \ldots, s_r$ be the children of the root of t. If P is a subset of $C_t = \{s_1, \ldots, s_r\}$, let t_P be the labeled tree formed by making the elements of P the children of a new root, keeping the original labels. Define the comultiplication by $\Delta(t) = \sum_{P \subseteq C_t} t_P \otimes t_{C_t \backslash P}$, where $X \backslash Y$ denotes the set-theoretic relative complement of Y in X. Define the counit by letting $\epsilon(t)$ be 1 if t has only one node (its root), and 0 otherwise. It can be shown that this makes $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ into a cocommutative coalgebra. We can define a grading on $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ by letting $k\{\mathcal{LT}(E_1, \ldots, E_M)\}_n$ be the subspace of $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ spanned by the trees with $n+1$ nodes.

A labeled ordered tree is a labeled tree for which there is a linear ordering of the children of each node. Denote the vector space over k with basis the set of labeled ordered trees by $k\{\mathcal{LOT}(E_1, \ldots, E_M)\}$. The definitions of the multiplication and comultiplication for $k\{\mathcal{LOT}(E_1, \ldots, E_M)\}$ are similar to the ones given above.

The following theorem is proved in [3].

Theorem 4.1 *The vector spaces $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ and $k\{\mathcal{LOT}(E_1, \ldots, E_M)\}$ are cocommutative graded connected Hopf algebras.*

The Milnor-Moore Theorem now says that we will know the structure of $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ once we know $P(k\{\mathcal{LT}(E_1, \ldots, E_M)\})$.

Theorem 4.2 *The set of labeled trees t whose root has exactly one child is a basis for $P(k\{\mathcal{LT}(E_1, \ldots, E_M)\})$.*

PROOF: It is immediate that any tree whose root has only one child is primitive. We now show that these trees span the primitive elements. Let $\mathcal{O} = k\{\mathcal{LT}(E_1, \ldots, E_M)\}$, and define a linear map $\pi : \mathcal{O} \otimes \mathcal{O} \to \mathcal{O}$ as follows: if t_1 and t_2 are labeled trees, let $\pi(t_1 \otimes t_2)$ be the labeled tree formed by identifying the roots of t_1 and t_2. In other words, $\pi(t_1 \otimes t_2)$ is the labeled

tree which has as subtrees of the root all the subtrees of the roots of t_1 and t_2. It is easy to see that if t is a labeled tree whose root has r children, then $\pi \circ \Delta(t) = 2^r t$. On the other hand, if $a = \sum a_t t \in P(\mathcal{O})$, we have that $\pi \circ \Delta(a) = 2a$. Since the trees t are linearly independent, it follows that $a_t = 0$ if the root of t has more than one child. This completes the proof of the theorem.

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 cocommutative Hopf algebra, and there is a Hopf 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. The following theorem is an immediate consequence of Theorem 5.1 of [3].

Theorem 4.3 *The map*

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

is injective.

Although we do not make use of it in this paper, we now describe briefly how these ideas apply to heap-ordered trees. We say that a rooted, finite tree is heap-ordered in case there is a total ordering on all nodes in the tree (called a key) 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 can be shown that $k\{\mathcal{LHOT}(E_1, \ldots, E_M)\}$ is also a cocommutative graded connected Hopf algebra using multiplication and comultiplication similar to those defined above. See [3] for details. As before, it follows immediately from Theorem 5.1 of [3] that the map

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

is injective. Here ϕ is defined defined by sending E_i to the heap-ordered tree with two nodes: the root with key 0 and its child labeled E_i with key 1. The map ϕ is then extended to all of $k<E_1, \ldots, E_M>$ using the fact that ϕ is an algebra homomorphism.

5 Simplification of higher order derivations

In this section we describe how labeled trees can be used to simplify the computation of differential operators. We begin by defining a map

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

as follows.

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 $1, \ldots, m$. We henceforth identify a node with the number assigned to it. To define the map, we make use of the summation indices $\mu_1, \ldots, \mu_m$, one associated with each node of the tree other than the root. Fix a node k of the tree t and let $l, \ldots, l'$ denote its children. Set

$$R(k; \mu_l, \ldots, \mu_{l'}) = \begin{cases} D_{\mu_l} \cdots D_{\mu_{l'}} a^{\mu_k}_{\gamma_k} & \text{if } k \text{ is not the root;} \\ D_{\mu_l} \cdots D_{\mu_{l'}} & \text{if } k \text{ is the root.} \end{cases}$$

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

Step 2. Define

$$\psi(t) = \sum_{)_1, \ldots,)_\mu{}^{m=1}}^{N} R(m) \cdots R(1)R(0)$$

and extend ψ to all $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ by 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 it is cheaper to compute ψ and ϕ together than to compute χ.

Proposition 5.1

i) *The map ψ is an algebra homomorphism.*

ii)

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

PROOF: The proof of (i) is a straightforward computation using Leibnitz' rule and is contained in [4]. Since χ and $\psi \circ \phi$ agree on the generating set $E_1, \ldots, E_M$, part (ii) follows from part (i). This completes the proof of the proposition.

In fact more is true: the map ψ respects the interaction of the comultiplication on $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ and the multiplication of R in the following sense:

Proposition 5.2 *For all $a, b \in R$, and for all $t \in k\{\mathcal{LT}(E_1, \ldots, E_M)\}$,*

$$((\psi \otimes \psi) \circ \Delta(t))\,(a \otimes b) = \psi(t)(ab).$$

PROOF: By the Milnor-Moore Theorem 3.6,

$$k\{\mathcal{LT}(E_1, \ldots, E_M)\} \cong U(P(k\{\mathcal{LT}(E_1, \ldots, E_M)\})).$$

Also, by Theorem 4.2, the Lie algebra $P(k\{\mathcal{LT}(E_1, \ldots, E_M)\})$ is isomorphic to the bialgebra generated by trees whose root has exactly one child. Therefore, trees whose root has exactly one child generate $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ as a k-algebra. Since ψ is an algebra homomorphism, we need only prove the assertion for trees whose root has only one child. Let t denote such a tree. Now

$$\Delta(t) = 1 \otimes t + t \otimes 1,$$

so that

$$(\psi \otimes \psi) \circ \Delta(t) = 1 \otimes \psi(t) + \psi(t) \otimes 1,$$

and

$$((\psi \otimes \psi) \circ \Delta(t))\,(a \otimes b) = a\,(\psi(t)b) + (\psi(t)a)\,b.$$

By Leibntz' rule, this is equal to $\psi(t)(ab)$, proving the proposition.

For many applications, it is important to know actions of these homomorphisms restricted to the Lie algebra generated by the derivations $E_1, \ldots, E_M$. The next proposition shows that if it is known that Lie algebra elements are being calculated, then only trees whose root has precisely one child need be considered.

Proposition 5.3

i) *Let $\mathcal{L}(E_1, \ldots, E_M))$ denote the Lie algebra generated by the derivations E_1, …, E_M. Then*

$$\begin{aligned} \phi(\mathcal{L}(E_1, \ldots, E_M)) &\subseteq P(k\{\mathcal{LT}(E_1, \ldots, E_M)\}) \\ \psi(P(k\{\mathcal{LT}(E_1, \ldots, E_M)\})) &\subseteq \operatorname{Der} R \end{aligned}$$

ii) *Elements in* $\mathcal{L}(E_1, \ldots, E_M)$ *are sent under* ϕ *to sums of trees whose root has only one child.*

PROOF: Since ϕ and ψ are Hopf algebra homomorphisms, they take primitive elements to primitive elements, which is (i). The primitive elements of the Hopf algebra $k<E_1, \ldots, E_M>$ are precisely the Lie algebra elements [9]. By Proposition 4.2, the primitive elements of $k\{\mathcal{LT}(E_1, \ldots, E_M)\}$ are those trees whose root has precisely one child. This proves (ii).

Let $\mathbf{Diff}(E_1, \ldots, E_M)$ denote the subalgebra of $\mathbf{Diff}(D_1, \ldots, D_N; R)$ generated by the derivations $E_1, \ldots, E_M$. To summarize, we have a commutative diagram

$$\begin{array}{ccc} k<E_1, \ldots, E_M> & \rightarrow & k\{\mathcal{LT}(E_1, \ldots, E_M)\} \\ & \searrow & \downarrow \\ & & \mathbf{Diff}(E_1, \ldots, E_M). \end{array}$$

As we shall see, in many common cases, it is more efficient to compute the composition of the arrow pointing to the right and arrow pointing down than to compute the arrow pointing southeast.

6 The cost of computing derivations

In this section, we analyse 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 an expression if we have several processors. This section is based upon the annoucement [5]. 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 differentiation 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 6.1 *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 differentiations and m multiplications (including the multiplications and differentiations involved in applying the operator $\chi(p)$).

Proposition 6.2 *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 [2]. 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 6.3 *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 6.4 *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 2.1 follows from Theorem 6.4.

Using Theorem 4.2, it is possible to construct even more efficient algorithms for computing the action of elements of $k<E_1, \ldots, E_M>$ which are known to be derivations: in this case, trees for which the root has more than one child need not even be constructed.

7 Computing derivations with several processors

In the previous sections, we showed how trees are naturally associated with the symbolic computation of higher order derivations. In this section, we show how trees lead to parallel algorithms for the symbolic computation of higher order derivations. Rather than try to state and prove the sharpest results, we state and prove what we feel are illustrative theorems. This section is based upon the announcement [6].

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. Let $\chi(p) \in \mathbf{Diff}(D_1, \ldots, D_N; R)$ denote the resulting linear differential operator. We 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 p_i is a monomial 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; hence, 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 viewed as an element of $k<E_1, \ldots, E_M>$ 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.

5. We assume that simultaneous writes to the memory belonging to some processor u by other processors are handled as if the other processors send messages to u requesting that the memory be written, and that the messages are queued in some fashion.

We establish some notation. Each monomial 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 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 monomial 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`$(u)(E_j)$, or more simply, `ProcessorList`(E_j);

2. an array `TermCount` containing counters such that `TermCount`$(u)[i]$ gives the number of times that monomial 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 $\mathcal{LT}_k(E_1, \ldots, E_M)$, 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 $E_{\gamma_{k+1}} \in \{E_1, \ldots, E_M\}$.

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. Consider the addition of `TermCount`$(u)[i]$ to `TermCount`$(u')[i]$ by processor u. (This can be thought of as processor u sending processor u' a message to increment `TermCount`$(u')[i]$ by `TermCount`$(u)[i]$.) The number of possible increments of `TermCount`$(u')[i]$, if u' is associated with a tree t' with $k+1$ nodes, is at most k. This is because one processor is associated with

```
(* 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.

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 $k+1$ 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.

The second remark is that 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 6.3. Therefore, given sufficiently many processors, $\psi(\sigma)$ can be computed in time $O(\log_2(m\,|\sigma|N^m))$.

Proposition 7.1 *The cost of computing $\phi(p)$ according to the algorithm given in Figure 1 is $O(l^2m^3)$.*

PROOF: Step 0, Step 1, and Step 3 take time $O(l)$. We give an estimate for Step 2. The outer loop is repeated $m-1$ times. The sequential loop contained in the "**do simultaneously**" loop is repeated l times. Since the length of `ProcessorList` is at most m, the sequential "**for**" loop contained in this 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 6.2, $\phi(p)$ can be computed in serial time $O(lm\,m!)$. Comparing this to the cost of the algorithm above shows

Theorem 7.2

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

References

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

[2] R. Grossman, *Evaluation of expressions involving higher order derivations,* Center For Pure and Applied Mathematics, PAM–367, University of California Berkeley.

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

[4] R. Grossman and R. G. Larson, *Solving Nonlinear Equations From Higher Order Derivations in Linear Stages,* Adv. in Math., to appear.

[5] R. Grossman and R. G. Larson, *Labeled Trees and the Algebra of Differential Operators,* "Algorithms and Graphs," R. Bruce Richter, Ed., AMS, 1989, pp. 81–87.

[6] R. Grossman and R. G. Larson, "Labeled trees and the efficient computation of derivations," in *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation,* ACM, New York, 1989, pp. 74–80.

[7] N. Jacobson, "Lie algebras," Interscience, New York, 1962.

[8] J. W. Milnor and J. C. Moore, *On the structure of Hopf algebras,* Ann. Math. (2) **81** (1965), 211–264.

[9] M. Sweedler, "Hopf Algebras," W. A. Benjamin, New York, 1969.

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