On the Performance of Antipode Algorithms for the Multivariable Output Feedback Hopf Algebra

Lance Berlin*, W. Steven Gray*, Luis A. Duffaut Espinosa†, Kurusch Ebrahimi-Fard‡

*Department of Electrical and Computer Engineering
Old Dominion University, Norfolk, VA 23529 USA

†Department of Electrical and Biomedical Engineering
University of Vermont, Burlington, VT 05405 USA

‡Department of Mathematical Sciences
Norwegian University of Science and Technology, NO-7491 Trondheim, Norway

***Abstract*—The feedback interconnection of two systems written in terms of Chen-Fliess series can be described explicitly in terms of the antipode of the output feedback Hopf algebra. At present, there are four known computational approaches to calculating this antipode. The main goal of this paper is to compare the computational performances of the three latest methods, two coproduct recursion methods and the derivation method, using a *Mathematica* implementation. The main conclusion is that the derivation method yields the best performance.**

***Keywords*—Nonlinear control systems, Chen-Fliess series, Hopf algebras, formal power series, symbolic computation**

I. INTRODUCTION

A convenient representation of a nonlinear input-output system is the *Chen-Fliess series* or *Fliess operator* [7], [18]. These functional series are described in terms of noncommutative formal power series, which provide a natural computational framework for describing system interconnections [8], [12], [15], identifying feedback invariants [9], [16], and performing system inversion [13], [14]. Of particular interest here is the feedback connection of two Fliess operators. The closed-loop system is known to always have a Fliess operator representation [4], and its corresponding generating series requires the computation of a compositional inverse. This inverse can be described and computed explicitly in the context of combinatorial Hopf algebras, namely in terms of the antipode of the *output feedback Hopf algebra* [8], [12]. In applications, it is necessary to calculate this antipode to the highest degree possible, but the computational complexity grows rapidly. At present, there are four known computational approaches. The first is to represent the underlying output feedback group in terms of a matrix group [10]. These infinite matrices are truncated and then symbolically inverted so that the antipode polynomials can be extracted. While this provides a straightforward software implementation, it was shown in [11] to be inefficient in comparison to other approaches. The next two techniques involve a natural recursive definition of the antipode based on its representation in terms of a geometric series [6]. These recursions are performed on the coproduct of the output feedback Hopf algebra, and there is a choice on which side of the coproduct to build the recursion. It turns out that the left recursion is known to create extra terms that later cancel out. So it has been long conjectured that this method is inferior to the right coproduct recursion that has been proved to be cancellation free [2]. The final and most recently developed method is based on derivations applied to the product in this Hopf algebra. This method was first observed to be implicit in the work of Devlin on the classical Poincaré center problem [3]. In a state space setting, it can be related to computing iterated Lie derivatives to determine series coefficients [18].

The main goal of this paper is to compare the computational performances of the two coproduct recursion methods and the derivation method for computing the output feedback Hopf antipode. To date, the only such study of this kind was the comparison between the matrix representation approach and the coproduct approaches in [11], and this was done only in the single-input, single-output (SISO) setting. Here the full multivariable versions of these algorithms will be tested using *Mathematica* implementations. Both wall time and resource consumption will be used as metrics for these algorithms. The main conclusion is that the derivation method yields the best performance.

The paper is organized as follows. In the next section some background information is briefly summarized. Then the antipode algorithms are described in the following section. In Section IV, a *Mathematica* implementation of the derivation algorithm is given. (Code from [11] with minor modifications for the multivariable case was used for the coproduct methods). Then the performance of these algorithms is documented in the subsequent section. The conclusions of the paper are provided in the final section.

II. BACKGROUND

A finite nonempty set of noncommuting symbols $X = \{x_0, x_1, \ldots, x_m\}$ is called an *alphabet*. Each element of X is called a *letter*, and any finite sequence of letters from X, $\eta = x_{i_1} \cdots x_{i_k}$, is called a *word* over X. The *length* of η, $|\eta|$, is the number of letters in η. Let $|\eta|_{x_i}$ denote the number of times the letter $x_i \in X$ appears in the word η. The set of all words including the empty word, $\emptyset$, is designated by X^*. It forms a monoid under catenation. Any mapping $c : X^* \to \mathbb{R}^\ell$ is called a *formal power series*. The value of c at $\eta \in X^*$ is written as (c, η) and called the *coefficient* of η in c. Typically, c is represented as the formal sum $c = \sum_{\eta \in X^*} (c, \eta)\eta$. The collection of all formal power series over X is denoted by $\mathbb{R}^\ell \langle\langle X \rangle\rangle$. It forms an associative $\mathbb{R}$-algebra under the catenation product and an associative and commutative $\mathbb{R}$-algebra under the *shuffle product*, that is, the bilinear product defined in terms of the shuffle product

978-1-5090-4780-2/17/$31.00

of two words

$$(x_i\eta) \sqcup (x_j\xi) = x_i(\eta \sqcup (x_j\xi)) + x_j((x_i\eta) \sqcup \xi),$$

where $x_i, x_j \in X$, $\eta, \xi \in X^*$ and with $\eta \sqcup \emptyset = \emptyset \sqcup \eta = \eta$ [7].

One can formally associate with any series $c \in \mathbb{R}^\ell\langle\langle X\rangle\rangle$ a causal m-input, ℓ-output operator, F_c, in the following manner. Let $\mathfrak{p} \geq 1$ and $t_0 < t_1$ be given. For a Lebesgue measurable function $u : [t_0, t_1] \to \mathbb{R}^m$, define $\|u\|_\mathfrak{p} = \max\{\|u_i\|_\mathfrak{p} : 1 \leq i \leq m\}$, where $\|u_i\|_\mathfrak{p}$ is the usual $L_\mathfrak{p}$-norm for a measurable real-valued function, u_i, defined on $[t_0, t_1]$. Let $L^m_\mathfrak{p}[t_0, t_1]$ denote the set of all measurable functions defined on $[t_0, t_1]$ having a finite $\|\cdot\|_\mathfrak{p}$ norm and $B^m_\mathfrak{p}(R)[t_0, t_1] := \{u \in L^m_\mathfrak{p}[t_0, t_1] : \|u\|_\mathfrak{p} \leq R\}$. Assume $C[t_0, t_1]$ is the subset of continuous functions in $L^m_1[t_0, t_1]$. Define inductively for each $\eta \in X^*$ the map $E_\eta : L^m_1[t_0, t_1] \to C[t_0, t_1]$ by setting $E_\emptyset[u] = 1$ and letting

$$E_{x_i\bar{\eta}}[u](t, t_0) = \int_{t_0}^{t} u_i(\tau) E_{\bar{\eta}}[u](\tau, t_0)\, d\tau,$$

where $x_i \in X$, $\bar{\eta} \in X^*$, and $u_0 = 1$. The input-output operator corresponding to c is the *Fliess operator*

$$F_c[u](t) = \sum_{\eta \in X^*} (c, \eta)\, E_\eta[u](t, t_0)$$

[7]. The operator is said to be of *Gevrey order* $s \in \mathbb{R}$ if there exists constants $K, M > 0$ such that

$$|(c, \eta)| \leq KM^{|\eta|}(|\eta|!)^s, \ \ \forall \eta \in X^*,$$

and s is the smallest number having this property [20]. If $s = 1$ then F_c constitutes a well defined mapping from $B^m_\mathfrak{p}(R)[t_0,\, t_0 + T]$ into $B^\ell_\mathfrak{q}(S)[t_0,\, t_0 + T]$ for sufficiently small $R, T > 0$, where the numbers $\mathfrak{p}, \mathfrak{q} \in [1, \infty]$ are conjugate exponents, i.e., $1/\mathfrak{p} + 1/\mathfrak{q} = 1$ [17]. (Here, $|z| := \max_{1 \leq i \leq \ell} |z_i|$ when $z \in \mathbb{R}^\ell$.) The set of all such *locally convergent* generating series is denoted by $\mathbb{R}^\ell_{LC}\langle\langle X\rangle\rangle$.

Given Fliess operators F_c and F_d, where $c, d \in \mathbb{R}^\ell\langle\langle X\rangle\rangle$, the parallel and product connections satisfy $F_c + F_d = F_{c+d}$ and $F_cF_d = F_{c \sqcup d}$, respectively [7]. When Fliess operators F_c and F_d with $c \in \mathbb{R}^\ell\langle\langle X\rangle\rangle$ and $d \in \mathbb{R}^m\langle\langle X\rangle\rangle$ are interconnected in a cascade fashion, the composite system $F_c \circ F_d$ has the Fliess operator representation $F_{c\circ d}$, where the *composition product* of c and d is given by

$$c \circ d = \sum_{\eta \in X^*} (c, \eta)\, \psi_d(\eta)(1)$$

[4], [5]. Here ψ_d is the continuous (in the ultrametric sense) algebra homomorphism from $\mathbb{R}\langle\langle X\rangle\rangle$ to the set of vector space endomorphisms on $\mathbb{R}\langle\langle X\rangle\rangle$, $\mathrm{End}(\mathbb{R}\langle\langle X\rangle\rangle)$, uniquely specified by $\psi_d(x_i\eta) = \psi_d(x_i) \circ \psi_d(\eta)$ with $\psi_d(x_i)(e) = x_0(d_i \sqcup e)$, $i = 0, 1, \ldots, m$ for any $e \in \mathbb{R}\langle\langle X\rangle\rangle$, and where d_i is the i-th component series of d ($d_0 := 1$). $\psi_d(\emptyset)$ is defined to be the identity map on $\mathbb{R}\langle\langle X\rangle\rangle$. This composition product is associative and $\mathbb{R}$-linear in its left argument.

When two Fliess operators F_c and F_d are interconnected to form a feedback system with F_c in the forward path and F_d in the feedback path, the generating series for the closed-loop system is denoted by the feedback product $c@d$. It can be computed explicitly using the Hopf algebra of coordinate functions associated with the underlying *output feedback group* [8], [12]. Specifically, define the set of operators $\mathscr{F}_\delta = \{I + F_c : c \in \mathbb{R}^m\langle\langle X\rangle\rangle\}$, where I denotes the identity map. It is convenient to introduce the symbol δ as the (fictitious) generating series for the identity map. That is, $F_\delta := I$ such that $I + F_c := F_{\delta+c} = F_{c_\delta}$ with $c_\delta := \delta + c$. The set of all such generating series for $\mathscr{F}_\delta$ will be denoted by $\mathbb{R}\langle\langle X_\delta\rangle\rangle$. The central idea is that $(\mathscr{F}_\delta, \circ, I)$ forms a group of operators under the composition

$$F_{c_\delta} \circ F_{d_\delta} = (I + F_c) \circ (I + F_d) = F_{c_\delta \circ d_\delta},$$

where $c_\delta \circ d_\delta := \delta + d + c\, \tilde{\circ}\, d$, and $\tilde{\circ}$ denotes the modified composition product (a variation of the composition product above, but with a direct feed term on the right-hand side [15]). The coordinate maps for the corresponding Hopf algebra H have the form

$$a^i_\eta : \mathbb{R}^m\langle\langle X\rangle\rangle \to \mathbb{R} : c \mapsto (c_i, \eta),$$

where $\eta \in X^*$ and $i = 1, 2, \ldots, m$. The commutative product is defined as

$$\mu : a^i_\eta \otimes a^j_\xi \mapsto a^i_\eta a^j_\xi,$$

where unit $\mathbf{1}$ is defined to map every c to zero. If the *degree* of a^i_η is defined as $\deg(a^i_\eta) = 2\,|\eta|_{x_0} + \sum_{j=1}^m |\eta|_{x_j} + 1$, then H is graded and connected with $H = \bigoplus_{n \geq 0} H_n$, where H_n is the set of all elements of degree n and $H_0 = \mathbb{R}\mathbf{1}$. The coproduct Δ is defined so that the formal power series product $c \circledcirc d := d + c\, \tilde{\circ}\, d$ for the group $\mathscr{F}_\delta$ satisfies

$$\Delta a^i_\eta(c, d) = a^i_\eta(c \circledcirc d) = (c_i \circledcirc d, \eta).$$

Of primary importance is the following lemma which describes how the group inverse $c^{-1}_\delta := \delta + c^{-1}$ is computed.

Lemma 2.1: [12] The Hopf algebra (H, μ, Δ) has an antipode S satisfying $a^i_\eta(c^{-1}) = (Sa^i_\eta)(c)$ for all $\eta \in X^*$ and $c \in \mathbb{R}^m\langle\langle X\rangle\rangle$.

The first few antipode terms are given below ordered by their degree:

$$\begin{aligned}
&H_1 : Sa^i_\emptyset = -a^i_\emptyset \\
&H_2 : Sa^i_{x_j} = -a^i_{x_j} \\
&H_3 : Sa^i_{x_0} = -a^i_{x_0} + a^i_{x_\ell}a^\ell_\emptyset \\
&H_3 : Sa^i_{x_jx_k} = -a^i_{x_jx_k} \\
&H_4 : Sa^i_{x_0x_j} = -a^i_{x_0x_j} + a^i_{x_\ell}a^\ell_{x_j} + a^i_{x_\ell x_j}a^\ell_\emptyset \\
&H_4 : Sa^i_{x_jx_0} = -a^i_{x_jx_0} + a^i_{x_jx_\ell}a^\ell_\emptyset \\
&H_4 : Sa^i_{x_jx_kx_l} = -a^i_{x_jx_kx_l} \\
&H_5 : Sa^i_{x_0^2} = -a^i_{x_0^2} + a^i_{x_\ell}a^\ell_{x_0} + a^i_{x_\ell x_0}a^\ell_\emptyset + a^i_{x_0x_\ell}a^\ell_\emptyset - \\
&\qquad\qquad a^i_{x_\nu}a^\nu_{x_\ell}a^\ell_\emptyset - a^i_{x_\nu x_\ell}a^\nu_\emptyset a^\ell_\emptyset,
\end{aligned}$$

where $i, j, k, l = 1, 2, \ldots m$[1].

The first theorem below illustrates how the antipode S is used to compute the feedback product. The subsequent result utilizes this concept for system inversion of SISO Fliess operators whose generating series have a well defined relative degree r in a certain sense [14]. In this case, the *natural part* of any $c \in \mathbb{R}\langle\langle X\rangle\rangle$ is given by $c_N = \sum_{k \geq 0}(c, x_0^k)x_0^k$,

[1]The Einstein summation notation is used throughout to indicate summations from either 0 or 1 to m, e.g., $\sum_{i=1}^m a_i b^i = a_i b^i$. It will be clear from the context which lower bound is applicable.

a series in $\mathbb{R}_{LC}[[X_0]]$, where $X_0 := \{x_0\}$. The multivariable version of this theorem appears in [13].

Theorem 2.1: [12] For any $c, d \in \mathbb{R}^m\langle\langle X\rangle\rangle$ it follows that $c@d = c \,\tilde{\circ}\, (-d \circ c)^{-1} = c \circ (\delta - d \circ c)^{-1}$.

Theorem 2.2: [14] Suppose $c \in \mathbb{R}\langle\langle X\rangle\rangle$ has relative degree r. Let y be analytic at $t = 0$ with generating series $c_y \in \mathbb{R}_{LC}[[X_0]]$ satisfying $(c_y, x_0^k) = (c, x_0^k)$, $k = 0, \ldots, r-1$. Then the input

$$u(t) = \sum_{k=0}^{\infty} (c_u, x_0^k) \frac{t^k}{k!},$$

where

$$c_u = \left(\left(\frac{(x_0^r)^{-1}(c - c_y)}{(x_0^{r-1}x_1)^{-1}(c)} \right)^{-1} \right)_N,$$

is the unique analytic solution to $F_c[u] = y$ on $[0, T]$ for some $T > 0$.

III. Antipode Algorithms

Three known methods for computing the antipode of the output feedback Hopf algebra are briefly described in this section. The first two methods employ coproducts directly associated with the coproduct Δ described above and what are called *left augmentation operators* [2], [12]. The third approach uses derivations on the product μ in combination with *right augmentation operators* [3].

A. Coproduct Methods

Define on the $\mathbb{R}$-vector space of coordinate functions with positive degree, V_+, the deshuffling coproduct $\Delta^j_{\sqcup\!\sqcup}(V_+) \subset V_+ \otimes V_+$:

$$\Delta^j_{\sqcup\!\sqcup}\, a^i_\emptyset = a^i_\emptyset \otimes a^j_\emptyset \tag{1a}$$

$$\Delta^j_{\sqcup\!\sqcup} \circ \theta_{x_k} = (\theta_{x_k} \otimes \mathrm{id} + \mathrm{id} \otimes \theta_{x_k}) \circ \Delta^j_{\sqcup\!\sqcup}, \tag{1b}$$

where id is the identity map on H, and θ_{x_k} is the left augmentation operator acting as an endomorphism on V_+ via $\theta_{x_k} a^i_\eta = a^i_{x_k\eta}$ for $k = 0, 1, \ldots, m$ and $i, j = 1, 2, \ldots, m$. In which case, the coproduct $\tilde{\Delta} := \Delta - \mathbf{1} \otimes \mathrm{id}$ can be computed by the following recursion.

Lemma 3.1: [12] The following identities hold:

(1) $\tilde{\Delta} a^i_\emptyset = a^i_\emptyset \otimes \mathbf{1}$
(2) $\tilde{\Delta} \circ \theta_{x_i} = (\theta_{x_i} \otimes \mathrm{id}) \circ \tilde{\Delta}$
(3) $\tilde{\Delta} \circ \theta_{x_0} = (\theta_{x_0} \otimes \mathrm{id}) \circ \tilde{\Delta} + (\theta_{x_i} \otimes \mu) \circ (\tilde{\Delta} \otimes \mathrm{id}) \circ \Delta^i_{\sqcup\!\sqcup}$

for $i = 1, 2, \ldots, m$.

The next result is a classical theorem from the theory of Hopf algebras.

Theorem 3.1: [6] The antipode, S, of any graded connected Hopf algebra (H, μ, Δ) can be computed for any $a \in H_k$, $k \geq 1$ by

$$Sa = -a - \sum (Sa'_{(1)}) a'_{(2)} = -a - \sum a'_{(1)} Sa'_{(2)},$$

where the reduced coproduct is $\Delta' a = \Delta a - a \otimes \mathbf{1} - \mathbf{1} \otimes a = \sum a'_{(1)} a'_{(2)}$ (using the notation of Sweedler).

Observe that in this recursion for S above there is a choice as to which side of the reduced coproduct one can apply S. So there are in fact two possible recursions here, a *left coproduct recursion* and a *right coproduct recursion*. While they are mathematically identical in that they give the same final result, there is significant differences in computation efficiency as will be discussed shortly. The *fully* recursive version of this algorithm for computing the antipode of the output feedback Hopf algebra is given below.

Theorem 3.2: [12] The antipode, S, of any $a^i_\eta \in V_+$ in the output feedback Hopf algebra can be computed by the following algorithm:

i. Recursively compute $\Delta^j_{\sqcup\!\sqcup}$ via (1).
ii. Recursively compute $\tilde{\Delta}$ via Lemma 3.1.
iii. Recursively compute S via Theorem 3.1 with $\Delta' a^i_\eta = \tilde{\Delta} a^i_\eta - a^i_\eta \otimes \mathbf{1}$.

It was observed in [11] for the SISO case that the left coproduct recursion generates terms that eventually cancel. It was later proved in [2] that the right coproduct recursion is always cancellation free, and therefore is significantly more efficient.

B. Derivation Method

Consider next the right augmentation operator

$$\tilde{\theta}_{x_i}(a^j_\eta) := a^j_{\eta x_i}$$

with $\tilde{\theta}_{x_i}(\mathbf{1}) := \mathbf{0}$. For any words $\eta, \xi \in X^*$ with $\eta := x_{i_1} x_{i_2} \cdots x_{i_k}$ observe that

$$\tilde{\theta}_\eta(a^j_\xi) := \tilde{\theta}_{x_{i_k}} \circ \cdots \circ \tilde{\theta}_{x_{i_2}} \circ \tilde{\theta}_{x_{i_1}}(a^j_\xi).$$

Moreover, the right-augment operator acts as a derivation (Leibniz rule) on products of coordinate functions, that is,

$$\tilde{\theta}_\eta(a^{j_1}_{\eta_1} a^{j_2}_{\eta_2} \cdots a^{j_k}_{\eta_k}) := \sum_{l=1}^{k} a^{j_1}_{\eta_1} a^{j_2}_{\eta_2} \cdots \tilde{\theta}_\eta(a^{j_l}_{\eta_l}) \cdots a^{j_k}_{\eta_k}.$$

Finally, define the operators:

$$\begin{aligned}
\tilde{\theta}'_{x_0}(a^j_\eta) &= -\tilde{\theta}_{x_0}(a^j_\eta) + \sum_{k=1}^{m} a^k_\emptyset \tilde{\theta}_{x_k}(a^j_\eta) \\
&= -a^j_{\eta x_0} + \sum_{k=1}^{m} a^k_\emptyset a^j_{\eta x_k} \\
\tilde{\theta}'_{x_i}(a^j_\eta) &= -\tilde{\theta}_{x_i}(a^j_\eta) = -a^j_{\eta x_i}, \quad i \neq 0 \\
\tilde{\Theta}'_\xi &= \tilde{\theta}'_{x_{i_k}} \circ \cdots \circ \tilde{\theta}'_{x_{i_2}} \circ \tilde{\theta}'_{x_{i_1}},
\end{aligned}$$

where $\xi := x_{i_1} x_{i_2} \cdots x_{i_k}$. Since these operators are defined linearly in terms of $\tilde{\theta}_{x_i}$ and $\tilde{\theta}'_{x_i}$, they also act as derivations on H. The following theorem describes how to compute the antipode, S, exclusively in terms of right-augmentation operators.

Theorem 3.3: [3] For any nonempty word $\eta \in X^*$, the antipode $S : H \to H$ in output feedback Hopf algebra can be written as

$$Sa^i_\eta = (-1)^{|\eta|-1} \tilde{\Theta}'_\eta(a^i_\emptyset).$$

Example 3.1: Let $X = \{x_0, x_1, x_2\}$ and $\eta = x_0x_1$. Then

$$\begin{aligned} Sa^2_\eta =& (-1)^{|\eta|-1}\tilde{\Theta}'_\eta(a^2_\emptyset) \\ =& (-1)^{2-1}\tilde{\theta}'_{x_1} \circ \tilde{\theta}'_{x_0}(a^2_\emptyset) \\ =& -\tilde{\theta}'_{x_1}(-a^2_{x_0} + a^1_\emptyset a^2_{x_1} + a^2_\emptyset a^2_{x_2}) \\ =& -a^2_{x_0x_1} + a^1_{x_1}a^2_{x_1} + a^1_\emptyset a^2_{x_1x_1} + a^2_{x_1}a^2_{x_2} + a^2_\emptyset a^2_{x_2x_1}. \end{aligned}$$

□

IV. Implementation in Mathematica

In this section an implementation in *Mathematica* is presented for the derivation method to compute the antipode of the output feedback Hopf algebra. Code from [11] was used for the coproduct methods by adding the internal summation in Lemma 3.1 (3) for the multivariable case. For all the methods, the noncommutative algebra is handled by the package *NCAlgebra* [19] invoked by

```
<<NC`
<<NCAlgebra`
```

In the SISO case, for example, the alphabet X is established with

```
X={x0,x1};
SetNonCommutative/@X;
```

Words are represented as noncommutative products such as

```
x0**x1**x0
```

for $x_0x_1x_0$. The double asterisk operation specifies noncommutative multiplication. The empty word, $\emptyset$, is treated here as the monomial $1\emptyset$, which through a slight abuse of notation is represented by simply 1. Coordinate functions are represented with the head A and two arguments: the first is the index i for the series c_i when $c \in \mathbb{R}^m\langle\langle X\rangle\rangle$, and the second is the associated word. For example,

```
A[2,x0**x1]
```

represents the coordinate function $a^2_{x_0x_1}$. For the SISO case, the first argument is always 1.

Functional definitions are now given for the implementation of the right-augment operator, $\tilde{\theta}_{x_i}$, denoted by RAug. The first argument is any polynomial expression of coordinate functions. The second is the index of the augmenting letter, $x_i \in X$. The last argument is for the alphabet X.

```
RAug[a_Plus,i_,x_List]:=
 Map[RAug[#,i,x]&,a]
RAug[a_Times,i_,x_List]:=
 RAug[a,i,x]=
  Sum[MapAt[RAug[#,i,x]&,
   a,ic], {ic,Length[a]}]
RAug[a_A^exp_,i_,x_List]:=
 exp a^(exp-1)*RAug[a,i,x]
RAug[a_A,i_,x_List]:=
 A[a[[1]],a[[2]]**x[[i+1]]]
RAug[a_,i_,x_List]:=0
```

The first of these definitions enforces the linearity of the operator over addition. The second is responsible for making the operator act as a derivation on products. The third definition is an optimized extension of this derivation rule for the cases where coordinate functions are repeated in a product. The fourth definition is the actual augmentation given by the definition $\tilde{\theta}_{x_i}(a^j_\eta) := a^j_{\eta x_i}$, while the last handles the special case $\tilde{\theta}_{x_i}(\mathbf{1}) := \mathbf{0}$. Note also that the second definition caches its result each time it is called in order to avoid repeating deep recursions that have already been calculated. As an example, $\tilde{\theta}_{x_2}(a^1_{x_0})$ on an alphabet $X = \{x_0, x_1, x_2\}$ can be computed by

```
RAug[A[1,x0],2,{x0,x1,x2}]
```

which returns the result $a^1_{x_0x_2}$ written in the form

```
A[1, x0**x2]
```

Next, the definitions for $\tilde{\theta}'_{x_i}$ are established. The arguments here are the same as for the RAug definitions.

```
RhoRAug[a_Plus,i_,x_List]:=
 Map[RhoRAug[#,i,x]&,a]
RhoRAug[a_,0,x_List]:=
 RhoRAug[a,0,x]=
  -RAug[a,0,x]+
  Sum[A[ic,1]*
    RAug[a,ic,x],
   {ic,1,Length[x]-1}]
RhoRAug[a_,i_,x_List]:=
 -RAug[a,i,x]
```

In the first definition, the linearity of the operator over addition is exercised. The second definition implements the rule $\tilde{\theta}'_{x_0}(a^j_\eta) = -\tilde{\theta}_{x_0}(a^j_\eta) + \sum_{k=1}^m a^k_\emptyset \tilde{\theta}_{x_k}(a^j_\eta) = -a^j_{\eta x_0} + \sum_{k=1}^m a^k_\emptyset a^j_{\eta x_k}$, while the third implements $\tilde{\theta}'_{x_i}(a^j_\eta) := -\tilde{\theta}_{x_i}(a^j_\eta) = -a^j_{\eta x_i}$, $i \neq 0$. Note here that there is no explicit rule given for $\tilde{\theta}'_{x_i}$ to act as a derivation. The derivation property is inherited algebraically as a consequence of $\tilde{\theta}'_{x_i}$ being defined linearly in terms of $\tilde{\theta}_{x_i}$. Just like the rules for the previous function, the second definition in this instance caches its result.

The final definitions compute the antipode of a given coordinate function a^i_η. The first argument is the coordinate function, and the second is the underlying alphabet X.

```
Antipode[A[s_,
  a_NonCommutativeMultiply],
  x_List]:=
 (-1)^(WordLength[a]-1)*
 Fold[RhoRAug[#1,
  FirstPosition[x,#2][[1]]-1,x]&,
 A[s,1],a]
Antipode[A[s_,1],x_List]:=-A[s,1]
Antipode[a_A,x_List]:=
 RhoRAug[A[a[[1]],1],
  FirstPosition[x,a[[2]]][[1]]-1,
   x]
```

The first definition directly implements Theorem 3.3, $Sa^j_\eta := (-1)^{|\eta|-1}\tilde{\Theta}'_\eta(a^j_\emptyset)$, with a functional composition of $\tilde{\theta}'_{x_i}$ operators. The second definition handles the specific

case of a coordinate function indexed by the empty word, $a^j_\emptyset$. The last case is for a coordinate function for a single letter word, that is, $a^j_{x_i}$. The antipode $Sa^2_{x_0x_1}$ can be calculated on the alphabet $X = \{x_0, x_1, x_2\}$ by

```
Antipode[A[2,x0**x1],{x0,x1,x2}]
```

which confirms the result given in Example 3.1, namely,

```
A[1,x1]A[2,x1]+A[2,x1]A[2,x2]
  -A[2,x0**x1]+A[1,1]A[2,x1**x1]
  +A[2,1]A[2,x2**x1]
```

V. Performance Analysis

In this section, a performance comparison between the derivation method and the two coproduct methods is given using the *Mathematica* implementations described above. The derivation and coproduct methods use very different data structures to represent the results. For this reason, more emphasis is placed here on the timing benchmarks as a metric for performance rather than on the spatial results. The latter mainly ensures that timing values are not inflated by memory saturation.

For consistency, all tests were run on the same Windows-based computer with a 2.70 GHz Intel Core i7-3740QM processor and 4×4 GB of 1600 MHz DDR3 SDRAM. *Mathematica* version 10.4 was used as well as excerpts of code from *NCAlgebra* version 4.0.6.

Two sets of tests were run for each implementation. One set examined performance for a given alphabet $X_S := \{x_0, x_1\}$ corresponding to a SISO system, while the other used the alphabet $X_M := \{x_0, x_1, x_2\}$ which models a two-input, two-output system. Within each set of tests, the degree of the coordinate function on which the antipode operation was applied was the independent variable. The worst case coordinate functions, $a^j_{x_0^i}$ where $\deg(a^j_{x_0^i}) = 2i + 1$, $i = 0, 1, \ldots$, were always used since they produce the most terms and recursions.

Each test was run independently on the same machine and without pre-cached results. Both the timing and memory data were obtained using *Mathematica's* built-in performance metric tools. The tests were run on successively increasing degrees of coordinate function until a particular case resulted in memory utilization exceeding 10 GB. Thus, in cases where no data is presented beyond a particular degree it is because that calculation could not be completed with the given memory restriction. No limit for execution time was imposed on the tests.

A. Timing Performance

First the execution times (wall times) of the antipode calculation are compared for each method. The SISO system performances with X_S are given in Figure 1, and the MIMO system performances with X_M are given in Figure 2. Table I provides the explicit timing results for the SISO case.

Note that in the SISO case the difference in execution times between the derivation and right coproduct methods decreases as the problem size increases with the derivation method giving slightly better performance. The left and right coproduct methods have roughly the same timings for degrees where there are relatively few cancellations in the left coproduct calculation (for example, there is one cancellation for degree five, 427 cancellations for degree 11 and 12,730 cancellations for degree 15 [2]). In the MIMO case, note that the derivation method consistently outperforms the other methods in terms of speed. In fact, this trend was observed to hold for all MIMO systems tested up to five letters [1].

TABLE I. Execution times (s) of antipode methods for X_S.

Degree	Derivation	Right Coproduct	Left Coproduct
1	0.000015	0.000133	0.000132
3	0.000131	0.000668	0.000573
5	0.000303	0.001243	0.002165
7	0.000918	0.002829	0.008646
9	0.003188	0.007377	0.045618
11	0.012189	0.020672	0.332750
13	0.045569	0.065496	2.937636
15	0.171627	0.221144	28.604094
17	0.683118	0.911619	312.933532
19	3.200415	3.779574	-
21	17.238338	20.226803	-
23	116.778690	132.333375	-
25	1028.466542	1069.589595	-

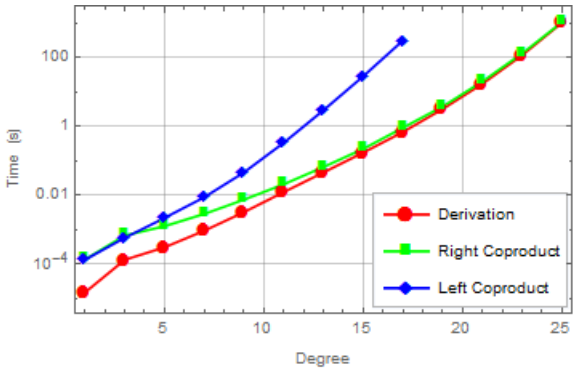

Fig. 1. Execution times of antipode methods for X_S.

B. Spatial Performance

Next the memory utilization of each method is presented. During the execution of each test, the total amount of system memory being used is monitored, and at the end of the calculation the peak usage is reported. This serves to reinforce the timing analysis by ensuring that the memory

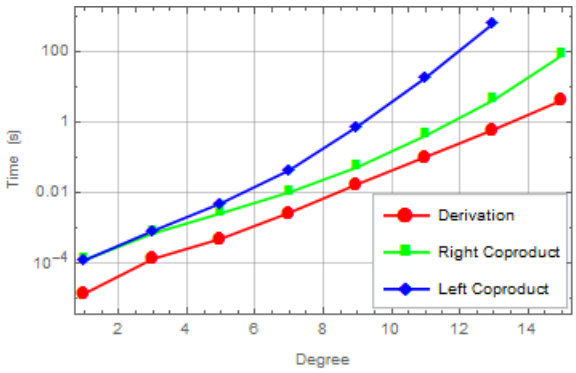

Fig. 2. Execution times of antipode methods for X_M.

resources of the system were not saturated during the tests, which would lead to an asymmetric increase in execution time. Additionally, one can identify from these results the consumption trends of each method and predict roughly for what size input the steepest performance decrease will begin to occur. The results for the SISO system tests are presented in Figure 3, while Figure 4 gives the results for the MIMO system.

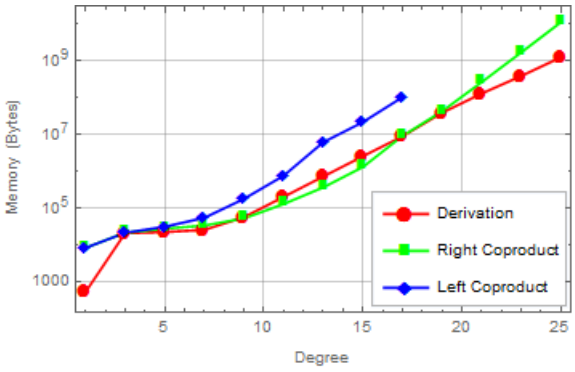

Fig. 3. Peak memory consumption of antipode methods for X_S.

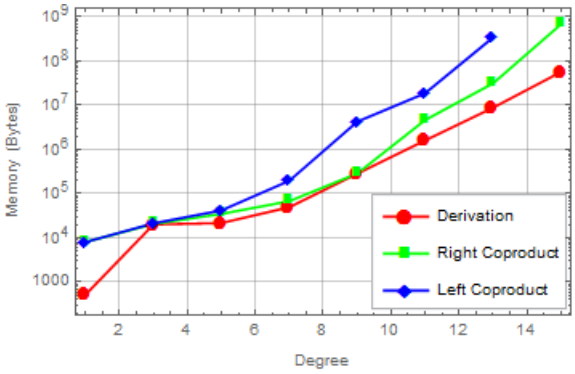

Fig. 4. Peak memory consumption of antipode methods for X_M.

Observe that both the derivation and right coproduct methods have similar growth trends in the SISO case up to coordinate functions of degree 19, after which the right coproduct method's consumption increases at a significantly faster rate. After degree 25, the right coproduct method was unable to complete the calculation under the memory constraint. The left coproduct method had the highest peak consumption in each SISO test, most likely due to the excessive inter-term cancellations that it generates. For the MIMO case, the derivation method not only utilizes fewer resources but grows at a notably slower rate than the others methods. Therefore, it is able to compute higher degree antipodes given the same memory constraints. The sudden jump in memory usage from degree 1 to degree 3 in the derivation method is a consequence of the implementation rather than the algorithm. Degree 1 corresponds to a transformation consisting of only of a sign change on the argument, while higher degree terms are calculated with a recursive method which involves considerably more overhead.

VI. CONCLUSIONS

Computational performances of the three most recent methods for computing the antipode of the output feedback Hopf algebra were compared using *Mathematica* implementations, namely two coproduct recursion methods and the derivation method. The main conclusion is that the derivation method yields superior performance in both execution time and memory usage, especially in the multivariable case.

REFERENCES

[1] L. N. Berlin, A Computer Algebra Implementation for Noncommutative Formal Power Series with Nonlinear Control Systems Applications, M.S. thesis, Old Dominion Univeristy, Norfolk, VA, to appear.

[2] L. A. Duffaut Espinosa, K. Ebrahimi-Fard, and W. S. Gray, A combinatorial Hopf algebra for nonlinear output feedback control systems, *J. Algebra*, 453 (2016) 609–643.

[3] K. Ebrahimi-Fard and W. S. Gray, Center Problem, Abel Equation and the Faà di Bruno Hopf Algebra for Output Feedback, *Int. Math. Res. Not.*, to appear, arXiv version: `http://lanl.arxiv.org/abs/1507.06939`.

[4] A. Ferfera, Combinatoire du Monoïde Libre Appliquée à la Composition et aux Variations de Certaines Fonctionnelles Issues de la Théorie des Systèmes, doctoral dissertation, University of Bordeaux I, 1979.

[5] ——, Combinatoire du monoïde libre et composition de certains systèmes non linéaires, *Astérisque*, 75-76 (1980) 87–93.

[6] H. Figueroa and J. M. Gracia-Bondía, Combinatorial Hopf algebras in quantum field theory I, *Rev. Math. Phys.*, 17 (2005) 881–976.

[7] M. Fliess, Fonctionnelles causales non linéaires et indéterminées non commutatives, *Bull. Soc. Math. France*, 109 (1981) 3–40.

[8] W. S. Gray and L. A. Duffaut Espinosa, A Faà di Bruno Hopf algebra for a group of Fliess operators with applications to feedback, *Systems Control Lett.*, 60 (2011) 441–449.

[9] ——, Feedback transformation group for nonlinear input-output systems, *Proc. 52nd IEEE Conf. on Decision and Control*, Florence, Italy, 2013, pp. 2570–2575.

[10] ——, A Faà di Bruno Hopf algebra for analytic nonlinear feedback control systems, in *Faà di Bruno Hopf Algebras, Dyson-Schwinger Equations, and Lie-Butcher Series*, K. Ebrahimi-Fard and F. Fauvet, Eds., IRMA Lectures in Mathematics and Theoretical Physics, Vol. 21, European Mathematical Society, Zürich, Switzerland, 2015, pp. 149-217.

[11] W. S. Gray, L. A. Duffaut Espinosa, and K. Ebrahimi-Fard, Recursive algorithm for the antipode in the SISO feedback product, *Proc. 21st Inter. Symp. on the Mathematical Theory of Networks and Systems*, Groningen, The Netherlands, 2014, pp. 1088–1093.

[12] ——, Faà di Bruno Hopf algebra of the output feedback group for multivariable Fliess operators, *Systems Control Lett.*, 74 (2014) 64–73.

[13] ——, Analytic left inversion of multivariable Lotka-Volterra models, *Proc. 54th IEEE Conf. on Decision and Control*, Osaka, Japan, 2015, pp. 6472–6477.

[14] W. S. Gray, L. A. Duffaut Espinosa, and M. Thitsa, Left inversion of analytic nonlinear SISO systems via formal power series methods, *Automatica*, 50 (2014) 2381–2388.

[15] W. S. Gray and Y. Li, Generating series for interconnected analytic nonlinear systems, *SIAM J. Control Optim.*, 44 (2005) 646–672.

[16] W. S. Gray, M. Thitsa, and L. A. Duffaut Espinosa Pre-Lie algebra characterization of SISO feedback invariants, *Proc. 53nd IEEE Conf. on Decision and Control*, Los Angeles, California, 2014, pp. 4807–4813.

[17] W. S. Gray and Y. Wang, Fliess operators on L_p spaces: Convergence and continuity, *Systems Control Lett.*, 46 (2002) 67–74.

[18] A. Isidori, *Nonlinear Control Systems*, 3rd Ed., Springer–Verlag, London, 1995.

[19] M. Stankus, J. W. Helton, M. de Oliveira, et al., *NCAlgebra*, Available: http://math.ucsd.edu/∼ncalg/. Accessed Jun. 2015.

[20] I. M. Winter-Arboleda, W. S. Gray, and L. A. Duffaut Espinosa, Fractional Fliess operators: Two approaches, *Proc. 49th Conf. Information Sciences and Systems*, Baltimore, MD, 2015.