21st International Symposium on
Mathematical Theory of Networks and Systems
July 7-11, 2014. Groningen, The Netherlands

Recursive Algorithm for the Antipode in the SISO Feedback Product

W. Steven Gray[1], Luis A. Duffaut Espinosa[2], and Kurusch Ebrahimi-Fard[3]

***Abstract*— Given two nonlinear input-output systems written in terms of Chen-Fliess functional expansions, the feedback interconnected system is in the same class with a generating series that can be computed explicitly via the antipode of a certain Faà di Bruno type Hopf algebra. This defines the feedback product of two generating series. Existing methods to compute this antipode are based on matrix inversion and are known to be inefficient. This paper has two objectives. The first is to use some recent advances in the area to produce a completely recursive algorithm for computing the antipode of the Hopf algebra for the SISO feedback group. The second objective is to provide a Mathematica implementation of the algorithm and evaluate its performance against the existing method using matrix inversion.**

***Index Terms*— Nonlinear control systems, formal power series, Chen-Fliess series, Hopf algebras, symbolic manipulation software**

***AMS Subject Classifications*—93C10, 16T05, 97N80**

I. Introduction

Given two nonlinear input-output systems written in terms of Chen-Fliess functional expansions [3], it was shown in [9], [11] that the feedback interconnected system is always well defined and in the same class. An explicit formula for the generating series of a single-input, single-output (SISO) closed-loop system was later provided in [6] using Hopf algebra methods. In particular, the so called *feedback product* of the two generating series for the component systems can be computed in terms of the antipode of a Faà di Bruno type Hopf algebra. In this case, the antipode was described in terms of a sequence of polynomials of increasing degree. While explicit, this somewhat brute force formula is not well-suited for software implementation. A more convenient approach is to represent the underlying feedback group in matrix form and then compute the antipode by matrix inversion [5], [7]. While easy to code, this approach is not efficient and limits the antipode calculation on conventional computers to about order seven in applications such as system inversion [8], [10].

Recently, it was shown in [4] that the Hopf algebra for the feedback group is connected under a grading that is distinct from the one described in [6]. This important observation means that a known *partially* recursive formula for the antipode can be applied [2]. This paper has two objectives. The first is to simply *connect the dots* and produce a *fully* recursive algorithm for computing the antipode of the Hopf algebra for the SISO feedback group. This involves carefully interweaving known results from [2], [4] and [14]. The second objective is to provide a Mathematica implementation of the recursive algorithm and evaluate its performance against the existing method which uses matrix inversion.

The paper is organized as follows. In the next section, some mathematical preliminaries are summarized. More complete treatments can be found in [6], [7], [9], [16]. In the subsequent section, the recursive antipode algorithm is developed. In Section IV, the Mathematica implementation is described. The paper's conclusions are given in the final section.

[1]W. Steven Gray is a visiting faculty member of the Instituto de Ciencias Matemáticas, Universidad Autónoma de Madrid, 28049 Madrid, Spain and on leave from Old Dominion University, Norfolk, Virginia 23529, USA, sgray@odu.edu

[2] Luis A. Duffaut Espinosa is an adjunct faculty member of the Department of Electrical and Computer Engineering, George Mason University, Fairfax, Virginia 22030, USA, lduffaut@gmu.edu

[3]Kurusch Ebrahimi-Fard is a Ramón y Cajal research fellow at the Instituto de Ciencias Matemáticas, Universidad Autónoma de Madrid, 28049 Madrid, Spain, kurusch@icmat.es

II. Preliminaries

Let $X = \{x_0, x_1\}$ be an alphabet, and X^* the set of all words over X including the empty word, $\emptyset$. The $\mathbb{R}$-vector space of formal power series over X with real coefficients is denoted by $\mathbb{R}\langle\langle X\rangle\rangle$. Two elementary products on $\mathbb{R}\langle\langle X\rangle\rangle$ are important in this work. The first is the *shuffle product*, which is the associative and commutative $\mathbb{R}$-bilinear product uniquely specified by the shuffle product of two words

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

where $x_i, x_j \in X$, $\eta, \xi \in X^*$ and with $\eta \sqcup\!\sqcup \emptyset = \eta$ [3], [14]. Consider its restriction to polynomials over X:

$$\mathrm{sh} : \mathbb{R}\langle X\rangle \otimes\ \mathbb{R}\langle X\rangle \to \mathbb{R}\langle X\rangle : p \otimes q \mapsto p \sqcup\!\sqcup q.$$

The corresponding adjoint map sh^* is the unique $\mathbb{R}$-linear map of the form $\mathbb{R}\langle X\rangle \to \mathbb{R}\langle X\rangle \otimes \mathbb{R}\langle X\rangle$ which satisfies the identity

$$(\mathrm{sh}(p \otimes q), r) = (p \otimes q, \mathrm{sh}^*(r))$$

for all $p, q, r \in \mathbb{R}\langle X\rangle$. The following theorem states an important duality.

Theorem 2.1: [14] The adjoint map sh^* is an $\mathbb{R}$-algebra morphism for the catenation product $\mathrm{cat} : p \otimes q \mapsto pq$. That is,

$$\mathrm{sh}^*(pq) = \mathrm{sh}^*(p)\,\mathrm{sh}^*(q)$$

for all $p, q \in \mathbb{R}\langle X\rangle$ with $\mathrm{sh}^*(1) = 1 \otimes 1$. In particular, for $x_i \in X$ and $\eta \in X^*$

$$\mathrm{sh}^*(x_i\eta) = (x_i \otimes 1 + 1 \otimes x_i)\mathrm{sh}^*(\eta). \quad (1)$$

The second relevant product on $\mathbb{R}\langle\langle X\rangle\rangle$ is the *modified composition product* [9]. Given two series $c, d \in \mathbb{R}\langle\langle X\rangle\rangle$, this product is defined as

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

where ϕ_d is the continuous (in the ultrametric sense) algebra homomorphism from $\mathbb{R}\langle\langle X\rangle\rangle$ to $\mathrm{End}(\mathbb{R}\langle\langle X\rangle\rangle)$ uniquely specified by $\phi_d(x_i\eta) = \phi_d(x_i) \circ \phi_d(\eta)$ with

$$\phi_d(x_0)(e) = x_0 e, \quad \phi_d(x_1)(e) = x_1 e + x_0(d \sqcup\!\sqcup e)$$

for any $e \in \mathbb{R}\langle\langle X\rangle\rangle$, and where $\phi_d(\emptyset)$ denotes the identity map on $\mathbb{R}\langle\langle X\rangle\rangle$.

ISBN: 978-90-367-6321-9

TABLE I
BASES IN THE GRADINGS OF $V=\bigoplus_{k\geq 0} V_k$ AND $H=\bigoplus_{k\geq 0} H_k$.

k	V_k	H_k	$\dim(V_k)$	$\dim(H_k)$
0	0	1	0	1
1	$a_\emptyset$	$a_\emptyset$	1	1
2	a_{x_1}	$a_{x_1}, a_\emptyset^2$	1	2
3	$a_{x_0}, a_{x_1^2}$	$a_{x_0}, a_{x_1^2}, a_\emptyset^3, a_\emptyset a_{x_1}$	2	4
4	$a_{x_0x_1}, a_{x_1x_0}, a_{x_1^3}$	$a_{x_0x_1}, a_{x_1x_0}, a_{x_1^3}, a_\emptyset^4,$ $a_\emptyset^2 a_{x_1}, a_\emptyset a_{x_0}, a_\emptyset a_{x_1^2}, a_{x_1}^2$	3	8
5	$a_{x_0^2}, a_{x_1x_0x_1}, a_{x_0x_1^2}, a_{x_1^2x_0}, a_{x_1^4}$	$a_{x_0^2}, a_{x_1x_0x_1}, a_{x_0x_1^2}, a_{x_1^2x_0}, a_{x_1^4},$ $a_\emptyset^5, a_\emptyset^3 a_{x_1}, a_\emptyset^2 a_{x_0}, a_\emptyset^2 a_{x_1^2}, a_\emptyset a_{x_0x_1},$ $a_\emptyset a_{x_1x_0}, a_\emptyset a_{x_1^3}, a_\emptyset a_{x_1}^2, a_{x_0}a_{x_1}, a_{x_1}a_{x_1^2}$	5	15

III. RECURSIVE ALGORITHM FOR ANTIPODE OF SISO FEEDBACK GROUP HOPF ALGEBRA

For any $\eta \in X^*$, the corresponding character map is the element of the dual space $\mathbb{R}\langle\langle X\rangle\rangle^*$

$$a_\eta : \mathbb{R}\langle\langle X\rangle\rangle \to \mathbb{R} : c \mapsto (c,\eta).$$

Let V denote the $\mathbb{R}$-vector space spanned by these maps. Consider also the free commutative $\mathbb{R}$-algebra, H, with product

$$\mu : a_\eta \otimes a_\xi \mapsto a_\eta a_\xi. \tag{2}$$

Three coproducts are now introduced. The first coproduct is $\Delta_{\sqcup\!\sqcup}(V) \subset V\otimes V$, which is isomorphic to sh^* via the character maps. That is,

$$\Delta_{\sqcup\!\sqcup} a_\emptyset = a_\emptyset \otimes a_\emptyset \tag{3a}$$
$$\Delta_{\sqcup\!\sqcup} \circ \theta_i = (\theta_i\otimes 1 + 1\otimes\theta_i)\circ\Delta_{\sqcup\!\sqcup}, \tag{3b}$$

where θ_i denotes the endomorphism on V specified by $\theta_i a_\eta = a_{x_i\eta}$, $i=0,1$.

Example 3.1: The first few terms of $\Delta_{\sqcup\!\sqcup}$ are:

$$\begin{aligned}
\Delta_{\sqcup\!\sqcup} a_\emptyset &= a_\emptyset\otimes a_\emptyset\\
\Delta_{\sqcup\!\sqcup} a_{x_{i_1}} &= a_{x_{i_1}}\otimes a_\emptyset + a_\emptyset\otimes a_{x_{i_1}}\\
\Delta_{\sqcup\!\sqcup} a_{x_{i_2}x_{i_1}} &= a_{x_{i_2}x_{i_1}}\otimes a_\emptyset + a_{x_{i_2}}\otimes a_{x_{i_1}} +\\
&\quad a_{x_{i_1}}\otimes a_{x_{i_2}} + a_\emptyset\otimes a_{x_{i_2}x_{i_1}}\\
\Delta_{\sqcup\!\sqcup} a_{x_{i_3}x_{i_2}x_{i_1}} &= a_{x_{i_3}x_{i_2}x_{i_1}}\otimes a_\emptyset + a_{x_{i_3}x_{i_2}}\otimes a_{x_{i_1}} +\\
&\quad a_{x_{i_3}x_{i_1}}\otimes a_{x_{i_2}} + a_{x_{i_3}}\otimes a_{x_{i_2}x_{i_1}} +\\
&\quad a_{x_{i_2}x_{i_1}}\otimes a_{x_{i_3}} + a_{x_{i_2}}\otimes a_{x_{i_3}x_{i_1}} +\\
&\quad a_{x_{i_1}}\otimes a_{x_{i_3}x_{i_2}} + a_\emptyset\otimes a_{x_{i_3}x_{i_2}x_{i_1}}.
\end{aligned}$$

□

The second coproduct is $\tilde\Delta(H)\subset V\otimes H$, which is induced by the identity

$$\tilde\Delta a_\eta(c,d) = a_\eta(c\,\tilde\circ\, d) = (c\,\tilde\circ\, d,\eta),\quad \eta\in X^*.$$

Let $|\eta|_{x_i}$ denote the number of times the letter $x_i\in X$ appears in the word η. If the *degree* of a_η is defined as $\deg(a_\eta) = 2\,|\eta|_{x_0} + |\eta|_{x_1} + 1$ and $\deg(a_\eta a_\xi) = \deg(a_\eta) + \deg(a_\xi)$, then V is a graded vector space, and (H,μ,Δ) is a graded and connected Hopf algebra, where the third coproduct is defined as $\Delta a_\eta = \tilde\Delta a_\eta + 1\otimes a_\eta$ [4]. Some bases in the gradings of $V=\bigoplus_{k\geq 0} V_k$ and $H=\bigoplus_{k\geq 0} H_k$, $H_0=\mathbb{R}$ are given in Table I. The coproduct $\tilde\Delta$ can be computed recursively as described next.

Lemma 3.1: [4] The following identities hold:

(1) $\tilde\Delta a_\emptyset = a_\emptyset\otimes 1$
(2) $\tilde\Delta\circ\theta_0 = (\theta_0\otimes\mathrm{id})\circ\tilde\Delta + (\theta_1\otimes\mu)\circ(\tilde\Delta\otimes\mathrm{id})\circ\Delta_{\sqcup\!\sqcup}$
(3) $\tilde\Delta\circ\theta_1 = (\theta_1\otimes\mathrm{id})\circ\tilde\Delta$,

where id denotes the identity map on H.

Example 3.2: Applying the identities in Lemma 3.1 gives examples of $\tilde\Delta a_\eta \in \bigoplus_{i+j=k} V_i\otimes H_j$ where $a_\eta\in V_k$:

$$\begin{aligned}
k=1 &: \tilde\Delta a_\emptyset = a_\emptyset\otimes 1\\
k=2 &: \tilde\Delta a_{x_1} = a_{x_1}\otimes 1\\
k=3 &: \tilde\Delta a_{x_0} = a_{x_0}\otimes 1 + a_{x_1}\otimes a_\emptyset\\
k=3 &: \tilde\Delta a_{x_1^2} = a_{x_1^2}\otimes 1\\
k=4 &: \tilde\Delta a_{x_0x_1} = a_{x_0x_1}\otimes 1 + a_{x_1}\otimes a_{x_1} + a_{x_1^2}\otimes a_\emptyset\\
k=4 &: \tilde\Delta a_{x_1x_0} = a_{x_1x_0}\otimes 1 + a_{x_1^2}\otimes a_\emptyset\\
k=4 &: \tilde\Delta a_{x_1^3} = a_{x_1^3}\otimes 1\\
k=5 &: \tilde\Delta a_{x_0^2} = a_{x_0^2}\otimes 1 + a_{x_1}\otimes a_{x_0} + a_{x_0x_1}\otimes a_\emptyset +\\
&\quad a_{x_1x_0}\otimes a_\emptyset + a_{x_1^2}\otimes a_\emptyset^2.
\end{aligned}$$

Note, in particular, that (1) ensures that the character $a_\emptyset$ does not appear on the left of any tensor product for coproducts of characters in V_k, $k\geq 2$. This fact provides a useful elementary property of the modified composition product given below. □

Lemma 3.2: Let $d\in\mathbb{R}\langle\langle X\rangle\rangle$ be fixed. Then $c\,\tilde\circ\,d = K\in\mathbb{R}$ if and only if $c=K$.[1]

Proof: The only non trivial assertion is that $c\,\tilde\circ\,d = K$ implies $c=K$. Clearly, if $c\,\tilde\circ\,d=K$ then $K = a_\emptyset(c\,\tilde\circ\,d) = \tilde\Delta a_\emptyset(c,d) = a_\emptyset c$. Furthermore, $0 = a_{x_1}(c\,\tilde\circ\,d) = \tilde\Delta a_{x_1}(c,d) = a_{x_1}c$. Now suppose $a_\eta c = 0$ for all $a_\eta\in V_k$ up to some fixed $k\geq 2$. Then for any $x_i\in X$

$$0 = \tilde\Delta a_{x_i\eta}(c,d) = a_{x_i\eta}c + \sum a_{(1)}c\,a_{(2)}d$$

[1] For notational convenience, $c = K\emptyset$ is written as $c=K$.

(in the notation of Sweedler [15]), where $a_{(1)} \neq a_\emptyset$. Therefore, $a_{x_i\eta}c = 0$. In which, case $c = K$. ■

Continuing now with the development of the main result, the next theorem states that for any graded connected Hopf algebra, the antipode can be computed by a *partially* recursive algorithm. That is, the coproduct needs to be computed first before the antipode recursion can be applied.

Theorem 3.1: [2] The antipode, α, of any graded connected Hopf algebra (H, μ, Δ) can be computed for any $a \in H_k$, $k \geq 1$ by

$$\alpha a = -a - \sum (\alpha a'_{(1)}) a'_{(2)}, \tag{4}$$

where the *reduced coproduct* is defined as $\Delta' a = \Delta a - a \otimes 1 - 1 \otimes a = \sum a'_{(1)} a'_{(2)}$. In addition, $\alpha(aa') = \alpha a' \alpha a$.

The next theorem provides a *fully* recursive algorithm to compute the antipode for the feedback group, which is the main theoretical result of the paper.

Theorem 3.2: The antipode, α, of any character $a_\eta \in V_k$ in the feedback product can be computed recursively by the following algorithm:

i. Recursively compute $\Delta_{\sqcup\!\sqcup}$ via (3).
ii. Recursively compute $\tilde{\Delta}$ via Lemma 3.1.
iii. Recursively compute α via Theorem 3.1 with $\Delta' a = \tilde{\Delta} a - a \otimes 1$.

Proof: In light of the previous results, the only detail is the minor observation that the feedback product is computed in terms of the Hopf algebra with coproduct $\Delta a = \tilde{\Delta} a + 1 \otimes a$ [4], [6]. In which case, the corresponding reduced coproduct is given by step iii. ■

Example 3.3: The first few elements of the reduced coproduct are:

$$
\begin{aligned}
k = 1 &: \Delta' a_\emptyset = 0 \\
k = 2 &: \Delta' a_{x_1} = 0 \\
k = 3 &: \Delta' a_{x_0} = a_{x_1} \otimes a_\emptyset \\
k = 3 &: \Delta' a_{x_1^2} = 0 \\
k = 4 &: \Delta' a_{x_0 x_1} = a_{x_1} \otimes a_{x_1} + a_{x_1^2} \otimes a_\emptyset \\
k = 4 &: \Delta' a_{x_1 x_0} = a_{x_1^2} \otimes a_\emptyset \\
k = 4 &: \Delta' a_{x_1^3} = 0 \\
k = 5 &: \Delta' a_{x_0^2} = a_{x_0 x_1} \otimes a_\emptyset + a_{x_1} \otimes a_{x_0} + a_{x_1 x_0} \otimes a_\emptyset + \\
&\qquad a_{x_1^2} \otimes a_\emptyset^2.
\end{aligned}
$$

Computing the antipode directly from (4) gives

$$
\begin{aligned}
H_1 &: \alpha a_\emptyset = -a_\emptyset \\
H_2 &: \alpha a_{x_1} = -a_{x_1} \\
H_3 &: \alpha a_{x_0} = -a_{x_0} + a_\emptyset a_{x_1} \\
H_3 &: \alpha a_{x_1^2} = -a_{x_1^2} \\
H_4 &: \alpha a_{x_0 x_1} = -a_{x_0 x_1} + a_{x_1}^2 + a_\emptyset a_{x_1^2} \\
H_4 &: \alpha a_{x_1 x_0} = -a_{x_1 x_0} + a_\emptyset a_{x_1^2} \\
H_4 &: \alpha a_{x_1^3} = -a_{x_1^3} \\
H_5 &: \alpha a_{x_0^2} = -a_{x_0^2} - (\alpha a_{x_0 x_1}) a_\emptyset - (\alpha a_{x_1}) a_{x_0} - \\
&\qquad (\alpha a_{x_1 x_0}) a_\emptyset - (\alpha a_{x_1^2}) a_\emptyset^2 \\
&\quad = -a_{x_0^2} - (-a_{x_0 x_1} + a_{x_1}^2 + a_\emptyset a_{x_1^2}) a_\emptyset - \\
&\qquad (-a_{x_1}) a_{x_0} - (-a_{x_1 x_0} + \cancel{a_\emptyset a_{x_1^2}}) a_\emptyset - \cancel{(-a_{x_1^2}) a_\emptyset^2} \\
&\quad = -a_{x_0^2} + a_\emptyset a_{x_0 x_1} - a_\emptyset a_{x_1}^2 - a_\emptyset^2 a_{x_1^2} + a_{x_0} a_{x_1} + \\
&\qquad a_\emptyset a_{x_1 x_0}.
\end{aligned}
$$

These are the known results computed in [6], [7] by brute force. The explicit calculations for $\alpha a_{x_0^2}$ are shown above to demonstrate that this approach, not unexpectedly, involves some inter-term cancelation. This is consistent with what is known about the classical Faà di Bruno Hopf algebra and the Zimmermann formula, which provides a cancelation free approach to computing the antipode [1], [12]. □

IV. MATHEMATICA IMPLEMENTATION OF RECURSIVE ANTIPODE FORMULA

In this section a Mathematica implementation of the recursive antipode algorithm in Theorem 3.1 is given. A comparison between this and the matrix inversion approach in [5], [7] is also presented. First an implementation of the bialgebra (H, μ, Δ) is described and then the antipode code is presented.

A. Bialgebra Implementation

Bialgebra implementation relies on the Mathematica package NCAlgebra to handle noncommutative algebra [13]. This package is loaded in Mathematica via the commands:

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

A character $a_\eta \in V$ with $\eta = x_{i_1} \cdots x_{i_n}$ is denoted in Mathematica by the symbol a$i_1 \cdots i_n$ where $i_j \in \{0, 1\}$ since $X = \{x_0, x_1\}$. In particular, $a_\emptyset$ is written as a. Characters up to degree 10 are constructed by:

```
degree=10;
cat01[w_]:={StringJoin[Flatten[Append[{w},{"0"}]]],
  StringJoin[Flatten[Append[{w},{"1"}]]]};
totcat01[w_]:=Flatten[Map[cat01,w],1];
coeffs:=ToExpression[Flatten[NestList[totcat01,{"a"},
  degree],1]];
```

Characters are set to be noncommutative in order to avoid undesirable behavior with respect to Mathematica's built-in symbolic tensor product:

```
SNC[coeffs];
NCSetOutput[All -> True];
```

The multiplication of symbolic tensors must satisfy

$$(a \otimes b)(c \otimes d) = a\,c \otimes b\,d$$

for $a, b, c, d \in H$. This product is defined by:

```
ProductTensor[(c1_:1)*(a1__⊗b1__),(c2_:1)*(a2__⊗b2__)]:=
    NCExpand[c1*c2*(a1**a2⊗b1**b2)];
ProductTensor[(c_:1)*(a_Plus),b_]:=c*ProductTensor[#,b]&/@a;
ProductTensor[a_,(c_:1)*(b_Plus)]:=c*ProductTensor[a,#]&/@b;
```

The product $**$ is used exclusively to multiply noncommutative symbols, while $*$ is used for commutative multiplication. The product (2) is implemented as follows:

```
MuProduct[tensor_]:=
    MapAt[Apply[NonCommutativeMultiply,#]&,tensor,
    Position[tensor,_⊗_]];
MuProduct[a_List]:=Apply[NonCommutativeMultiply,a];
MuProduct[a_⊗"1"]:=a;
MuProduct["1"⊗a_]:=a;
MuProduct[{a_,"1"}]:=a;
MuProduct[{"1",a_}]:=a;
```

This function accepts two types of arguments:

```
In[1]:= MuProduct[a⊗b]

Out[1]= a**b

In[2]:= MuProduct[a,b]

Out[2]= a**b
```

Mathematica and NCAlgebra supply many built-in routines to expand or simplify expressions. For example, **NCExpand** and **TensorExpand** expand expressions with respect to addition/multiplication and the symbolic tensor product, respectively. These routines, however, apply many expansion rules that are unnecessary in the present context. As a consequence, the computation time increases dramatically when computing antipodes of order 6 and higher as discovered in [8], [10], The next function supplies only those expansions and simplifications needed for the routines presented here:

```
DistributeTensor[(c_: 1)*tensor_]:=
      Distribute[c*Distribute[tensor]];
DistributeTensor[tensor_Plus]:=
      DistributeTensor[#]&/@tensor;
```

The endomorphism θ_i is defined by:

```
ThetaMap[(c_: 1)*a_, num_]:=
  c*ToExpression[StringTake[ToString[a],1]<>ToString[num]<>
      StringDrop[ToString[a],1]];
```

The left shift of a character is provided by:

```
ThetaMapShift[(c_: 1)*a_,num_]:=c*ToExpression[
  If[StringTake[ToString[a],{2}]===ToString[num],
    ToExpression[StringReplacePart[ToString[a],"",
    {2,2}]],0]];
ThetaMapShift[0,num_]:=0;
```

A convenient fact to exploit in calculations is that most generating series seldom have full support, that is, many of their coefficients are zero. The next function plays a key role in handling zero coefficients in the antipode computation:

```
IdentityReplace[a_,rules_]:=Identity[a]/.rules;
```

In Lemma 3.1 this function must be applied only to the right argument id of the tensor products since $\theta_i a_\eta(c) = a_{x_i\eta}(c)$ is not necessarily zero when $a_\eta(c) = 0$.

Three functions over tensors are needed for the computation of the shuffle and modified composition coproducts:

$$(f \otimes \mathrm{id})(a_\eta \otimes a_\xi) = f(a_\eta) \otimes a_\xi \tag{5a}$$

$$(\mathrm{id} \otimes g)(a_\eta \otimes a_\xi) = a_\eta \otimes g(a_\xi) \tag{5b}$$

$$(f \otimes g)(a_\eta \otimes a_\xi) = f(a_\eta) \otimes g(a_\xi). \tag{5c}$$

The first two are implemented by:

```
TensorComposition[f_,tensor_,side_]:=
  DistributeTensor[MapAt[f[#]&,tensor,
    Map[Append[#,side]&,Position[tensor,_⊗_]]]];
```

where (5a) is obtained when side=1 and (5b) when side=2. This function produces the output:

```
In[3]:= TensorComposition[f,a0⊗a1,1]
In[4]:= TensorComposition[f,a0⊗a1,2]

Out[3]= f[a0]⊗a1
Out[4]= a0⊗f[a1]
```

The implementation for (5c) is given by:

```
TensorComposition2[f_,g_,tensor_]:=
  DistributeTensor[MapAt[f[#1]⊗g[#2]& @@ #&,
    tensor,Position[tensor,_⊗_]]];
```

with corresponding output:

```
In[5]:= TensorComposition2[f,g,a0⊗a1]

Out[5]= f[a0]⊗g[a1]
```

The only reason for differentiating between the three functions above is to speed up the computations. In principle, one could use **TensorComposition2** exclusively, however, it is preferable to avoid applying extra functions even in the case where one of the functions is the identity function.

The next function considers tensors of order three as arguments. The need for such a function is apparent in the second summand of Lemma 3.1(2). Thus,

```
TensorCompositionMu[f_,g_,tensor_] :=
  DistributeTensor[MapAt[f[#1]⊗g[List[##2]]& @@ # &,
    tensor,Position[tensor,_⊗_]]];
```

implements, for example,

$$(f \otimes g)(a_{x_0} \otimes a_{x_1} \otimes a_{x_0^2}) = f(a_{x_0}) \otimes g(a_{x_1}, a_{x_0^2}),$$

which is equivalent to

```
In[6]:= TensorCompositionMu[f,g,a0⊗a1⊗a00,]

Out[6]= f[a0]⊗g[a1,a00]
```

To compute the coproduct $\tilde{\Delta}$ one also needs to implement

$$(\theta_i \otimes \mu)(a_\eta \otimes a_\xi \otimes a_\nu) = \theta_i(a_\eta) \otimes a_\xi a_\nu.$$

A function central to implementing (4) is a variation of **TensorComposition**:

```
MuRightTensor[f_,tensor_,side_]:=
  Distribute[MuProduct[TensorComposition[f,tensor,side]]];
MuRightTensor[f_,0]:=0;
```

This function implements

$$\mu((f \otimes \mathrm{id})(a_\eta \otimes a_\xi)) = f(a_\eta)\, a_\xi.$$

Note that **DistributeTensor** is used here in order to avoid unnecessary and time consuming expansions.

The shuffle coproduct is implemented as follows:

```
CoproductShuffle[a_] :=
  Module[{s1,s2},s1=ToString[a];If[StringLength[s1]==1,
    If[NumericQ[a],$Failed,ToExpression[a]⊗ToExpression[a]],
    s2=ToExpression[StringTake[s1,{2,2}]];
    TensorComposition[ThetaMap[#,s2]&,
      CoproductShuffle[ThetaMapShift[s1,s2]],1]+
     TensorComposition[ThetaMap[#,s2]&,
      CoproductShuffle[ThetaMapShift[s1,s2]],2]]];
```

This function is consistent with Example 3.1:

```
In[7]:=  CoproductShuffle[a]
In[8]:=  CoproductShuffle[a1]
In[9]:=  CoproductShuffle[a0]
In[10]:= CoproductShuffle[a00]
In[11]:= CoproductShuffle[a01]
In[12]:= CoproductShuffle[a10]
In[13]:= CoproductShuffle[a11]

Out[7]=  a⊗a
Out[8]=  a⊗a1+a1⊗a
Out[9]=  a⊗a0+a0⊗a
Out[10]= a⊗a00+2a0⊗a0+a00⊗a
Out[11]= a⊗a01+a0⊗a1+a01⊗a+a1⊗a0
Out[12]= a⊗a10+a0⊗a1+a1⊗a0+a10⊗a
Out[13]= a⊗a11+2a1⊗a1+a11⊗a
```

Coefficients and words are related via the characters. It is useful to introduce a symbol for the empty word, $\emptyset$. Its interaction with respect to the noncommutative product in NCAlgebra is specified as follows:

```
SNC[∅,x0,x1];
NonCommutativeMultiply[∅,a__]:=a;
NonCommutativeMultiply[a__,∅]:=a;
NonCommutativeMultiply[∅,∅]:=∅;
```

The first line above defines $\emptyset$ and the letters of X to be noncommutative for convenience, but the subsequent set of rules establishes the commutativity of $\emptyset$ precisely with the letters of X. Functions mapping words to their characters and vice versa are given below:

```
WordsToCoeffsAux[a_]:=
  If[WordLength[a]==0,"",
    StringTake[ToString[FirstLetter[a]],{2,2}]<>
    WordsToCoeffsAux[LeftShift[a,FirstLetter[a]]]];
WordsToCoeffs[a_,label_]:=
  ToExpression[ToString[label]<>WordsToCoeffsAux[a]];

CoeffsToWords[a_]:=
  If[StringLength[ToString[a]]==1,∅,
   ToExpression["x"<>StringTake[ToString[a],{2,2}]
    <>"**"<>ToString[CoeffsToWords[ThetaMapShift[a,
    StringTake[ToString[a],{2,2}]]]]]];
CoeffsToWords[a_Plus]:=CoeffsToWords[#]&/@a;
CoeffsToWords[0]:=0;
```

For example,

```
In[14]:= WordsToCoeffs[x0**x1**x0**x1,a]
In[15]:= CoeffsToWords[a0101]

Out[14]= a0101
Out[15]= x0**x1**x0**x1
```

To compute the shuffle coproduct, sh^*, the action of the function **CoeffsToWords** with respect to the tensor product and multiplication by reals has to be made explicit:

```
CoeffsToWords[a_⊗b_]:=CoeffsToWords[a]⊗CoeffsToWords[b];
CoeffsToWords[(c_: 1)*a_⊗b_]:=c*CoeffsToWords[a⊗b];
```

Thus, the shuffle coproduct for words is computed from the shuffle coproduct of characters by:

```
CoproductShuffleWord[a_]:=
   CoeffsToWords[CoproductShuffle[WordsToCoeffs[a,v]]]; ,
```

This provides an implementation of (1). For example,

```
In[16]:= CoproductShuffleWord[x0**x1]

Out[16]= ∅⊗x0**x1+x0⊗x1+x0**x1⊗∅+x1⊗x0
```

The reduced shuffle coproduct of words is easily implemented as:

```
ReducedCoproductShuffleWord[a_]:=
  CoproductShuffleWord[a]−∅⊗a−a⊗∅;
```

The modified composition coproduct, $\tilde{\Delta}$, is implemented via Lemma 3.1 by:

```
CoproductComposition[a_,rules_:1]:=
  Module[{s1,s2,s3},s1=ToString[a]; If[StringLength[s1]==1,
   If[NumericQ[a],$Failed,ToExpression[a]⊗"1"],
    s2=ToExpression[StringTake[s1,{2,2}]]; If[rules===1,
     s3=TensorComposition[ThetaMap[#,s2]&,
      CoproductComposition[ThetaMapShift[s1,s2],rules],1];
     If[IntegerQ[s2],If[s2==0,s3+
      TensorCompositionMu[ThetaMap[#,1]&,
       MuProduct,TensorComposition[
        CoproductComposition[#,rules]&,
         CoproductShuffle[ThetaMapShift[s1,s2]],1]],
      If[s2==1,s3]]],
     s3=TensorComposition2[ThetaMap[#,s2]&,
      IdentityReplace[#,rules]&,
      CoproductComposition[ThetaMapShift[s1,s2],rules]];
     If[IntegerQ[s2],If[s2==0,s3+
      TensorCompositionMu[ThetaMap[#,1]&,MuProduct,
       TensorComposition2[CoproductComposition[#,rules]&,
        IdentityReplace[#,rules]&,
        CoproductShuffle[ThetaMapShift[s1,s2]]]],
      If[s2==1,s3]]]]]];
```

This gives, for example,

```
In[22]:= CoproductComposition[a]
In[23]:= CoproductComposition[a1]
In[24]:= CoproductComposition[a0]
In[25]:= CoproductComposition[a00]
In[26]:= CoproductComposition[a01]
In[27]:= CoproductComposition[a10]
In[28]:= CoproductComposition[a11]

Out[22]= a⊗1
Out[23]= a1⊗1
Out[24]= a0⊗1+a1⊗a
Out[25]= a00⊗1+a01⊗a+a1⊗a0+a10⊗a+a11⊗a**a
Out[26]= a01⊗1+a1⊗a1+a11⊗a
Out[27]= a10⊗1+a11⊗a
Out[28]= a11⊗1
```

which is consistent with Example 3.2. Observe that the **NonCommutativeMultiply** command does not simplify expressions such as a∗∗a into a^2. This was done purposely to avoid the appearance of unexpected patterns at the output of functions. The modified composition reduced coproduct is given by

```
ReducedCoproductComposition[a_,rules_:1]:=
  CoproductComposition[a,rules]−TensorProduct[a,"1"];
```

B. Antipode Implementation

An implementation of the recursive antipode algorithm in Theorem 3.1 is describe in this subsection. But first the tools provided in the previous subsection are tested by the following implementation of the antipode of the Hopf algebra $(\mathbb{R}\langle X\rangle, \mathrm{cat}, \mathrm{sh}^*)$, denoted by $\alpha_{\sqcup\!\sqcup}$:

```
ShuffleAntipode[a_]:=
  −a−MuRightTensor[ShuffleAntipode,
    ReducedCoproductShuffleWord[a],1];
ShuffleAntipode[0]:=0;
ShuffleAntipode[∅]:=∅;
```

Observe as an example:

```
In[17]:= ShuffleAntipode[x0]
In[18]:= ShuffleAntipode[x0**x1]
In[19]:= ShuffleAntipode[x0**x1**x0]
In[20]:= ShuffleAntipode[x1**x1**x0**x1]
In[21]:= ShuffleAntipode[x1**x0**x1**x1**x1]

Out[17]= −x0
Out[18]= x1**x0
Out[19]= −x0**x1**x0
Out[20]= x1**x0**x1**x1
Out[21]= −x1**x1**x1**x0**x1
```

These results agree with the known formula

$$\alpha_{\sqcup\!\sqcup}\,\eta = (-1)^{|\eta|}\sigma(\eta),$$

where $\sigma(x_{i_1}x_{i_2}\cdots x_{i_n}) := x_{i_n}x_{i_{n-1}}\cdots x_{i_1}$ [14]. The antipode for (H,μ,Δ) is implemented in a similar fashion by

```
CompositionAntipode[a_,rules_:1]:=
  −a−MuRightTensor[CompositionAntipode[#,rules]&,
    ReducedCoproductComposition[a,rules],1];
```

In which case:

```
In[29]:= CompositionAntipode[a]
In[30]:= CompositionAntipode[a1]
In[31]:= CompositionAntipode[a0]
In[32]:= CompositionAntipode[a00]
In[33]:= CompositionAntipode[a01]
In[34]:= CompositionAntipode[a10]
In[35]:= CompositionAntipode[a11]

Out[29]= −a
Out[30]= −a1
Out[31]= −a0+a1**a
Out[32]= −a00+a01**a+a1**a0+a10**a−a1**a1**a−a11**a**a
Out[33]= −a01+a1**a1+a11**a
Out[34]= −a10+a11**a
Out[35]= −a11
```

This agrees with the results in Example 3.3.

Table II and Figure 1 compare the run times of the matrix based antipode algorithm described in [5], [7] and the recursive algorithm given in Theorem 3.1. As expected, the recursive algorithm dramatically outperforms the matrix approach. In particular, the matrix approach could not compute the antipode of a0000000 before running out of memory. When look-up tables where employed to reduce memory usage, the run time for this case was still over 24 hours. On the other hand, the recursive algorithm required approximately three minutes to compute the antipode of this particular character.

Table III and Figure 2 show the run times for the computation of the antipodes for a series having full support (all

TABLE II

RUN TIMES IN SECONDS TO COMPUTE THE COMPOSITION ANTIPODE VIA THE MATRIX APPROACH AND THE RECURSIVE ALGORITHM

order	matrix approach	recursive algorithm
1	0.146361	0.000603
2	0.164789	0.010459
3	0.145437	0.043076
4	0.511338	0.267987
5	13.039	1.845762
6	7476.04	15.626257
7	–	196.302212
8	–	2270.566715

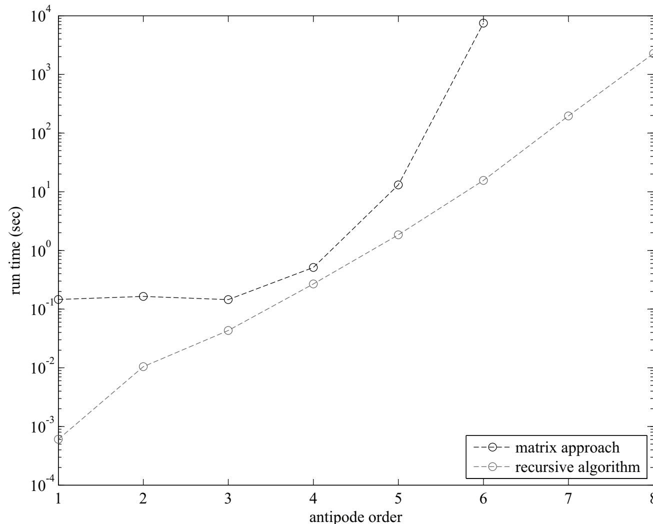

Fig. 1. Run time in seconds versus composition antipode order via the matrix approach and the recursive algorithm.

coefficients are nonzero) and a proper series (the coefficient of the empty word is assumed to be zero) via the recursive algorithm. This single property, which is easy to exploit in the recursive approach, greatly reduced the run time. The matrix approach, on the other hand, can not be so readily adapted to take advantage of such structure. So the topic was not pursued here.

V. CONCLUSIONS

Recent advances in understanding the combinatoric nature of the modified composition product provide a completely recursive algorithm for computing the antipode of the Hopf algebra for the SISO feedback group. This algorithm was provided and then implemented in Mathematica. Its performance against a known method based on matrix inversion was evaluated and shown to provide significant improvement.

TABLE III

RUN TIMES IN SECONDS TO COMPUTE THE COMPOSITION ANTIPODE FOR A SERIES WITH FULL SUPPORT AND A PROPER SERIES VIA THE RECURSIVE ALGORITHM

order	full support	proper series
1	0.000603	–
2	0.010459	0.000204
3	0.043076	0.000413
4	0.267987	0.001348
5	1.845762	0.016633
6	15.626257	0.346429
7	196.302212	1.521978
8	2270.566715	6.011801
9	–	33.695830
10	–	194.971646

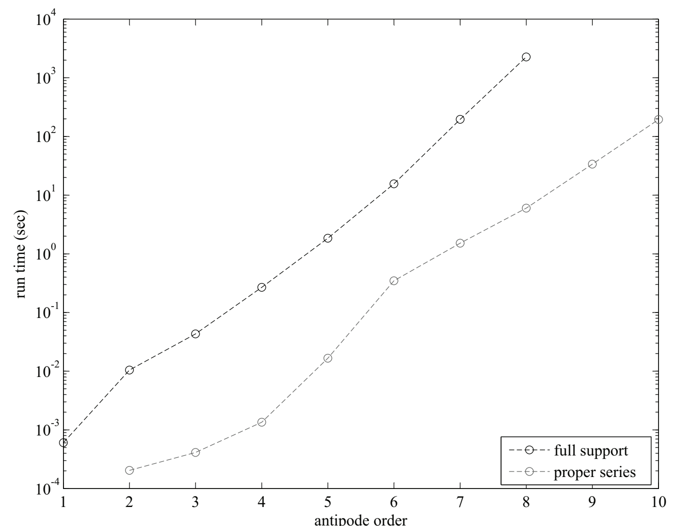

Fig. 2. Run time in seconds versus the composition antipode order for a series with full support and a proper series via the recursive algorithm.

REFERENCES

[1] H. Einziger, Incidence Hopf Algebras: Antipodes, Forest Formulas, and Noncrossing Partitions, Doctoral dissertation, The George Washington University, Washington, DC, 2010.

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

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

[4] L. Foissy, The Hopf algebra of Fliess operators and its dual pre-Lie algebra, `http://arxiv.org/abs/1304.1726v2`, 2013.

[5] A. Frabetti, Renormalization Hopf algebras and combinatorial groups, `http://arxiv.org/abs/0805.4385v2`, 2009.

[6] 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.

[7] W. S. Gray and L. A. Duffaut Espinosa, 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 Lect. Math. Theor. Phys., Eur. Math. Soc., Strasbourg, France, 2014, to appear.

[8] W. S. Gray, L. A. Duffaut Espinosa, and M. Thitsa, Left inversion of analytic nonlinear SISO systems via formal power series methods, *Automatica*, under review.

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

[10] W. S. Gray, M. Thitsa, and L. A. Duffaut Espinosa, Left inversion of analytic SISO systems via formal power series methods, *Proc. 15th Latin American Control Conf.*, Lima, Peru, 2012.

[11] W. S. Gray and Y. Wang, Formal Fliess operators with applications to feedback interconnections, *Proc. 18th Inter. Symp. Mathematical Theory of Networks and Systems*, Blacksburg, Virginia, 2008.

[12] M. Haiman and W. Schmitt, Incidence algebra antipodes and Lagrange inversion in one and several variables, *J. Combin. Theory Ser. A*, 50 (1989) 172–185.

[13] The NCAlgebra Suite, Version 4.0, available at `http://math.ucsd.edu/~ncalg`, 2012.

[14] C. Reutenauer, *Free Lie Algebras*, Oxford University Press, New York, 1993.

[15] M. E. Sweedler, *Hopf Algebras*, W. A. Benjamin, Inc., New York, 1969.

[16] M. Thitsa and W. S. Gray, On the radius of convergence of interconnected analytic nonlinear input-output systems, *SIAM J. Control Optim.*, 50 (2012) 2786–2813.