The Music of Streams

M. Douglas McIlroy

Dartmouth College, Hanover, New Hampshire 03755, USA

Abstract

Data streams make preeminent instruments for playing certain classical themes from analysis. Complex networks of processes, effortlessly orchestrated by lazy evaluation, can enumerate terms of formal power series ad infinitum. Expressed in a language like Haskell, working programs for power-series operations are tiny gems, because the natural programming style for data streams fits the mathematics so well—often better than time-honored summation notation.

> The cleverest copyist is the one whose music is performed with the most ease without the performer guessing why. — Jean Jacques Rousseau, *Dictionary of Music*

1 Overture

Like persistent folk tunes, the themes I intend to play here have been arranged for various ensembles over many years. Some performances have been angular, and some melodic, but all share the staying power of good music. The themes stick in mind, to be enjoyed again and again as each performance exposes new surfaces and depths.

My subject is "power-stream compositions," a phrase intended to suggest a discipline of remarkable expressiveness and style, as well as compositionality. Algorithms on power streams are succinct, beautiful, and inevitable in their fitness for the purpose. As motivating examples for stream-based computing, power streams are compelling in the same way that quicksort is for recursion.

I shall show power-stream compositions in two styles, with the unifying motif of lazy lists. Power series are represented as sequences of coefficients. Operators on series acquire values in order from their inputs, compute result values in order, and deliver them as output. Input is never received ahead of the needs of the output. Such operators are obviously compositional.

In the first style, *Horner form,* a power series

$$\begin{aligned} f(z) &= f_0 + f_1 z + f_2\, z^2 + f_3\, z^3 + \cdots \\ &= f_0 + z(f_1 + z(f_2 + z(f_3 + ...))) \end{aligned}$$

is viewed recursively as a head term f_0 and a tail power series $\overline{f}$ that satisfy $f(z) = f_0 + z\overline{f}(z)$. The series f is represented by the list $[f_0, f_1, f_2, f_3, \cdots]$.

In the second style, *Maclaurin form,* a power series

$$f(z) = f(0) + f'(0)z + f''(0)z^2/2! + f^{(3)}(0)z^3/3! + \cdots$$

is viewed as a head term $f(0)$ and a tail series, f', of the same form as f. The tail is the derivative of f and satisfies

$$f(z) = f(0) + \int_0^z f'(t)dt.$$

The series f is represented by the list $[f(0), f'(0), f''(0), f^{(3)}(0), \cdots]$.

The list $[1, 1, 1, 1, \cdots]$ interpreted in Horner form is the geometric series $1 + z + z^2 + z^3 + \cdots$. The same list interpreted in Maclaurin form is the exponential series $1 + z + z^2/2! + z^3/3! + \cdots$.

The lazy-list formulation relegates concerns about sequencing and storage management to an evaluating engine. For example, in programming the usual convolution formula for the product of power series,

$$\check{}\,(\textstyle\sum_{i=0}^{\infty} f_i z^i) \times (\sum_{i=0}^{\infty} g_i z^i) = \sum_{i=0}^{\infty} z^i \sum_{j=0}^{j=i} f_j g_{i-j},$$

one must keep book on a tangle of subscripts and on ever-growing arrays of input coefficients. In the lazy list realization of Horner form, the program becomes a smooth one-liner:

$$(f_0 : \overline{f}) \times g@(g_0 : \overline{g}) = f_0 g_0 : f_0 \times \overline{g} + g \times \overline{f}$$

The operators (:) and (@), taken from Haskell (Peterson and Hammond 1997), denote the list constructor and the "as" operator: $g@(g_0 : \overline{g})$ means operand g has structure $(g_0 : \overline{g})$.

1.1 Conventions

Polynomials may be represented as finite power series.

Operations on power series are defined by rules that tell how to compute the head of the result and how recursively to compute the tail. In applying a rule, the first subrule whose patterns match the operands is used. The anonymous identifier _ names unused quantities. When a scalar operand occurs where a list is expected, the scalar is converted to a one-element list of the expected type. Thus, when f is a list of integers, $1 + f$ becomes $[1] + f$.

The rules are formal. There is no concern with analytic convergence or computational divergence. Nor is there a guarantee that results will be uniquely represented; any result may end in an arbitrary sequence of zeros. However, any result given by the rules will agree with the analytically correct result when such exists.

2 First theme: Horner form

A series $f(z) = f_0 + f_1 z + f_2 z^2 + \cdots$ may be viewed as a head f_0 and a tail series $\overline{f}$, so that

$f(z) = f_0 + z\,\overline{f}(z),$

with the list representation

$f = f_0 : \overline{f}.$

2.1 Arithmetic

$$
\begin{array}{ll}
\textbf{Negative.} & -[\,] = [\,] \\
& -(f_0 : \overline{f}) = -f_0 \;:\; -\overline{f} \\
\textbf{Sum.} & f + [\,] = f \\
& [\,] + f = f \\
& (f_0 : \overline{f}) + (g_0 : \overline{g}) = f_0{+}g_0 \;:\; \overline{f}{+}\overline{g} \\
\textbf{Product.} & [\,] \times \_ = [\,] \\
& \_ \times [\,] = [\,] \\
& (f_0 : \overline{f}) \times g@(g_0 : \overline{g}) = f_0 g_0 \;:\; f_0 \times \overline{g} + \overline{f} \times g \\
\textbf{Quotient.} & [\,] \,/\, [\,] = \textbf{error} \\
& [\,] \,/\, (0 : \_) = \textbf{error} \\
& [\,] \,/\, \_ = [\,] \\
& (0 : \overline{f}) \,/\, (0 : \overline{g}) = \overline{f} \,/\, \overline{g} \\
& (f_0 : \overline{f}) \,/\, g@(g_0 : \overline{g}) = \textbf{ let } q_0 = f_0/g_0 \textbf{ in } q_0 \;:\; (\overline{f} - q_0 \times \overline{g})/g
\end{array}
$$

To verify the product rule, write the series in head-tail style, then expand the product into a form that can be read in head-tail style:

$f \times g = (f_0 + z\overline{f}) \times (g_0 + z\overline{g}) = f_0 g_0 + z(f_0 \times \overline{g} + \overline{f} \times g).$

The last quotient rule expresses long division; it may be checked by multiplying through by g.

Difference and nonnegative integer power may be defined in the usual way in terms of negative, sum and product.

2.2 Composition and reversion

Expanding the composition $f \circ g$ of power series f and g in head-tail style gives

$$\begin{aligned} f(g) &= f_0 + g \times \overline{f}(g) \\ &= f + (g_0 + z\overline{g}) \times \overline{f}(g) \\ &= (f_0 + g_0\overline{f}(g)) + z\overline{g} \times \overline{f}(g). \end{aligned}$$

In general, the the head element of the output list cannot be calculated in finite time: unfolding the parenthesized subexpression leads to an infinite sum for the head element. We can proceed, however, when $g_0 = 0$ (second rule below), or when f is a polynomial (third rule).

Composition. $[\,] \circ \_ = [\,]$

$$(f_0 : \overline{f}) \circ g@(0 : \overline{g}) = f_0 : \overline{g} \times (\overline{f} \circ \overline{g})$$

$$(f_0 : \overline{f}) \circ g@(g_0 : \overline{g}) = (f_0 + g_0\overline{f}(g)) + (0 \;:\; \overline{g} \times \overline{f}(g))$$

Reversion. revert $(0 : \overline{f}) = r$ **where** $r = 0 \;:\; 1/(\overline{f} \circ r)$

The latter rule, for reversion (functional inversion) of power series, may be developed by expanding the identity $f(r(z)) = z$, where $r(z)$ is the inverse series and $f(0) = 0$.

The reversion rule is a tribute to the lucidity of power streams. Reversion is sufficiently tricky to have inspired a considerable literature, rife with subscripts. One of the more concise treatments (Knuth, 1969) takes half a page of pseudocode to describe an algorithm equivalent to this working one-liner. The restriction to operands with head term zero may be relaxed for polynomial operands, as it was for composition.

2.3 Calculus

Let D be the differentiation operator. Since $D(f_0 + z\overline{f}) = \overline{f} + z(D\overline{f})$, we have

Derivative (slow). $D(\_ : \overline{f}) = \overline{f} + (0 : D\overline{f})$

$$D\_ = [\,]$$

This formulation of the derivative takes $O(n^2)$ coefficient-domain additions to compute n terms of the result. Trading away elegance, we may obtain an $O(n)$ algorithm by counting terms and using the fact that $Dz^n = nz^{n-1}$. The integral, defined by $\int_0^z f(t)dt$, may also be calculated by counting terms. (I do not know of a "slow integral" rule that hides the counting.)

Fast derivative. $D[\,] = [\,]$

$D(\_ : \overline{f}) = (h\,1\,\overline{f})$ **where**

$h\ \_\ [\,] = [\,]$

$h\ n\ (g_0 : \overline{g}) = ng_0\ :\ (h\ (n+1)\ \overline{g})$

Fast integral. $\int [\,] = [\,]$

$\int f = 0\ :\ (h\,1\,f)$ **where**

$h\ \_\ [\,] = [\,]$

$h\ n\ (g_0 : \overline{g}) = g_0/n\ :\ (h\ (n+1)\ \overline{g})$

Example 1, elementary functions. Sine, cosine and exponential may be defined as solutions of initial value problems,

$D\sin(x) = \cos(x), \quad \sin(0) = 0$

$D\cos(x) = -\sin(x), \quad \cos(0) = 1$

$D\exp(x) = \exp(x), \quad \exp(0) = 1$

Integration converts the differential equations to a neat program:

$\sin = \int \cos$

$\cos = 1 - \int \sin$

$\exp = 1 + \int \exp$

Example 2, fast exponential. Calculation of n terms of $\exp(f(z))$, where $f(0) = 0$, by the program $\exp \circ f$ takes $O(n^3)$ coefficient-domain operations (Knuth, 1969). Gilles Kahn exploited the fact that $\exp(f)$ is a solution of $y' = yf'$ with $y(0) = 1$ to get an $O(n^2)$ method:

$\exp \circ f = y$ **where** $y = 1 + \int(y \times Df)$.

Example 3, square root. A power series f with constant term 1 has a square root q, of the same form, that satisfies $q^2 = f$. Hence

$2qDq = Df$

$Dq = Df/2q$

$q = 1 + \int Df/2q$

from which follows the program

sqrt $f@(1 : \_) = q$ **where** $q = 1 + \int(Df/(2q))$.

Example 4, tough tests. Classical identities provide a trove of tests of power-series code. A simple test that exercises every basic operation is to check an initial segment of the series T defined by

$t = [0, 1]$

$\mathrm{arctan} = \int(1/(1 + t^2))$

$\tan = \mathrm{revert}\ \mathrm{arctan}$

$T = \tan - \sin / \cos$

Every term of T should vanish.

2.4 Variation: multivariate series

Coefficients in a power series may have any type for which the usual arithmetic functions are defined and finite and for which multiplication is commutative. In particular coefficients may be univariate polynomials, in which case the series is said to be in *nested form.*

Example 5, Pascal triangle. The generating function for the binomial coefficients of order n is $(1 + y)^n$. To enumerate the binomial coefficients for all n, we may replace z in the geometric series $1+z+z^2+z^3+\cdots$ by $(1+y)z$. That is, we compose the generating function for the geometric series, $1/(1-z)$, with the polynomial $0 + (1 + y)z$:

$(1/[1, -1]) \circ [[0], [1, 1]]$

The result is the Pascal triangle,

$[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], \cdots]$.

2.4.1 Homogeneous form

It is often convenient to group terms of a bivariate power series in x and y by total degree, with the nth group being a homogeneous polynomial of degree n. The groups may be carried as coefficients in a power series in z:

$f(x, y, z) = (f_{00}) + (f_{10}x + f_{01}y)z + (f_{20}x^2 + f_{11}xy + f_{02}y^2)z^2 + \cdots$

Since the degree of any variable in a term is determinable from the degree of the other two, we can elide any one variable—i.e. set the variable to 1. Since elision commutes with addition and multiplication, there is a direct isomorphism between the ring of nested-form series $f(1, y, z)$ and the ring of homogeneous-form series $f(x, y, 1)$. We can exploit this isomorphism to compute with homogeneous-form series.

The following rules take partial derivatives of a homogeneous-form series. The y-derivative of the homogeneous polynomial $f_1(x, y)$ is calculated by applying the ordinary derivative operator to the isomorph $f_1(1, y)$. The x-derivative works on the isomorph $f_1(x, 1)$, whose terms come in reverse order. So that the first term that operator D sees is the term of x-degree zero, f_1 must have exactly $n+1$ terms, where n is the total degree of f_1.

Partial derivative $D_y(\_ : \overline{f}@(f_1 : \_)) = (D\ f_1)\ :\ (D_y\ \overline{f})$

$D_y\,\_ = [\,]$

$D_x(\_ : \overline{f}@(f_1 : \_)) = (\text{reverse} \cdot D \cdot \text{reverse}) f_1\ :\ (D_x\ \overline{f})$

$D_x\,\_ = [\,]$

The operator $(\cdot)$ is ordinary functional composition. It is distinguished from $(\circ)$, which applies to sequences, not functions.

Example 6, more tough tests. The identities $e^{x+y} = e^x e^y$ and $D_x e^{x+y} = e^{x+y}$ yield test functions, T_1 and T_2, that should evaluate to zero.

$x = [[0],\ [1, 0]]$

$y = [[0],\ [0, 1]]$

$T_1 = (\exp \circ (x + y)) - (\exp \circ x) \times (\exp \circ y)$

$T_2 = D_x(\exp \circ (x + y)) - D_y(\exp \circ (x + y))$

These tests are truly vigorous exercises that take $O(n^5)$ rational operations to carry out to n terms.

3 Second theme: Maclaurin form

In Maclaurin form, a power series is represented by the list of derivatives evaluated at the origin:

$[f(0), f'(0), f''(0), f^{(3)}(0), \cdots]$.

The representation of the derivative f' is the tail of the representation of f. Writing $f_0 = f(0)$, we have

$f = f_0 \underline{:} f'$

Operator $(\underline{:})$ is the ordinary list constructor, decorated as a reminder that it now constructs Maclaurin form.

Derivative and integral operators shift sequences left or right. Some arithmetic rules and the reversion rule are the same as for Horner form. Only distinctive rules are given here. In particular, rules for empty lists are omitted. The rules for arithmetic and composition are direct consequences of the usual formulas for derivative of product and quotient, and the chain rule, $(f(g))' = f'(g)g'$.

Derivative. $D(\_ \underline{:} f') = f'$

Integral. $\int f = 0 \underline{:} f$

Product. $f@(f_0 : f') \times g@(g_0 : g') = f_0 g_0 \underline{:} (f \times g' + g \times f')$

Quotient. $f@(f_0 \underline{:} f') \,/\, g@(g_0 \underline{:} g') = q$ **where** $q = f_0/g_0 \underline{:} (f' - q \times g')/g$

Composition. $(f_0 \underline{:} f') \circ g@(0 \underline{:} g') = f_0 \underline{:} (f' \circ g) \times g'$

Conversion from Horner to Maclaurin form (hm) and vice versa (mh) are straightforward.

Conversions. $\mathrm{hm}[\,] = [\,]$

$\mathrm{hm}(f@(f_0 : \_)) = f_0 \underline{:} \mathrm{hm}(Df)$

$\mathrm{mh}[\,] = [\,]$

$\mathrm{mh}(f_0 \underline{:} f') = f_0 + \int \mathrm{mh}(f')$

3.1 Behavioral differential equations

Relations written in the form $y = y_0 \underline{:} y'$ are called *behavioral differential equations* (Rutten, 1999), or BDEs.

Example 7, elementary functions. From the differential equations for the elementary functions (Example 1) we read off these BDEs:

$\sin = 0 \underline{:} \cos$

$\cos = 1 \underline{:} -\sin$

$\exp = 1 \underline{:} \exp$

Executing any of these functions—a trivial exercise—yields a sequence of derivatives evaluated at the origin. The sequence for the cosine is $\check{\ }[1, 0, -1, 0, 1, 0, -1, 0, \cdots]$.

3.2 Doing math

The power-stream idea, though born in computing, is fruitful in mathematics, too. Derivations of mathematical relationships among objects defined by BDEs can be quite direct and elegant.

Example 8, reciprocal of the exponential. Verify the identity $1/e^x = e^{-x}$, knowing only that e^x denotes the solution of $\exp = 1 \underline{:} \exp$. Let

$$
\begin{aligned}
q &= 1/\exp \\
&= 1 \mathbin{\underline{:}} 1' - q\exp' / \exp && \text{Quotient rule} \\
&= 1 \mathbin{\underline{:}} -q\exp / \exp && \text{Example 7, Derivative rule} \\
&= 1 \mathbin{\underline{:}} -q \\
z &= [0,1] \\
r &= \exp\circ(-z) \\
&= (1 \mathbin{\underline{:}} \exp)\circ(-z) && \text{Example 7} \\
&= 1 \mathbin{\underline{:}} (\exp'\circ(-z)) \times (-z)' && \text{Composition rule} \\
&= 1 \mathbin{\underline{:}} (\exp\circ(-z)) \times (-1) && \text{Example 7, Derivative rule} \\
&= 1 \mathbin{\underline{:}} -r
\end{aligned}
$$

Since q and r satisfy the same behavioral differential equation, the desired result holds. Notice that the calculation deals with infinite objects without using sums or limits. It instead uses properties of the fixed points of BDEs.

Example 9, Bell numbers. The numbers B_n count distinct ways to distribute n distinguishable objects among indistinguishable boxes (Bell 1934). They have an exponential generating function

$$B(z) = \sum_{n=0}^{\infty} B_n z^n/n! = e^{e^z-1}$$

Thus the Bell numbers are produced by the program

$$
\begin{aligned}
\text{bell} &= \exp\circ(\exp - 1) \\
&= \exp\circ((1 \mathbin{\underline{:}} \exp) - 1) && \text{Example 7} \\
&= \exp\circ(0 \mathbin{\underline{:}} \exp)
\end{aligned}
$$

It is a pleasant exercise to show that

$$\text{bell} = 1 \mathbin{\underline{:}} \text{bell} \times \exp$$

is equivalent; both programs produce

$$[1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \cdots].$$

4 Coda

4.1 Complexity

Efficiency can vary dramatically with algorithm. Section (2.3) gives $O(n^2)$ and $O(n)$ rules for the derivative. Example 2 gives $O(n^3)$ and $O(n^2)$ algorithms for calculating the exponential of a series. More dramatically, the product operators for Maclaurin and Horner forms respectively take $O(2^n)$ and $O(n^2)$ coefficient-domain operations to compute n terms. Evidently it would be advantageous to compute the product of Maclaurin-form series by bypassing (Melzak 1983) to Horner form and back. On the other hand, it takes $O(n)$ coefficient-domain operations to compute n terms of the elementary functions in either form.

4.2 Power streams as system tests

Besides exercising the basic rules, the "tough tests" of Examples 4 and 6 also severely stress implementations of lazy-list languages, and of stream-processing systems similar to CSP (Hoare, 1985). Example 4 alone has shown up bugs or severe capacity limitations in several systems and languages in which I have tried them. Fortunately, all were promptly fixed by responsive implementers.

4.3 Real Haskell

While the programs above deviate from pure Haskell only in minor typographic ways, they omit details of operator overloading for the sake of accessibility. The full story of how to make the code work in Haskell is given by McIlroy (1999).

4.4 Influences

Power-stream compositions, once mainly an oral tradition, have begun to take their place in the literature, Two remarkable papers, by Karczmarczuk (1997) and by Pavlović and Escardó (1998), exploit the two forms. Karczmarczuk deals mainly with special functions of mathematical physics in Horner form. Pavlović and Escardó examine Maclaurin form and related representations from an algebraic standpoint, and advocate power streams as a productive formalism for calculus. McIlroy discusses Horner form as an application of Haskell, and Hehner (1993) integrates the subject with the logic of programs. Rutten (1999) relates Maclaurin form to automata theory.

4.5 Resolution

As the music of power streams, always fascinating, has been arranged for successively more agile instruments, the melodic form has become purer and more alluring. Early performances of power streams resembled concerts of street cries, in which transactions were shouted out among producers and consumers: “Get me a term, oh/ Here is your term, oh”—a striking but stolid form. Scored now for a well tuned organ of lazy evaluation, the genre has, I trust, become worthy of performance for that prince of computing, Edsger W. Dijkstra.

References

[1] Bell, E. T. 1934. Exponential numbers. *American Mathematical Monthly* 41: 411-419.

[2] Hehner, E. C. R. 1993. *A Practical Theory of Programming*, 143-148. Springer.

[3] Hoare, C. A. R. 1985. *Communicating Sequential Processes.* Prentice-Hall.

[4] Karczmarczuk, J. 1997. Generating power of lazy semantics. *Theoretical Computer Science*, 9: 203–219.

[5] Knuth, D. E. 1969. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, §4.7. Addison-Wesley.

[6] McIlroy, M. D. 1999. Functional pearl: power series, power serious. *Journal of Functional Programming*, 9: 323–335; also at `http://www.cs.dartmouth.edu/~doug/haskell/pearl.ps.gz`.

[7] Melzak, Z. A. 1983. *Bypasses: a Simple Approach to Complexity.* John Wiley & Sons.

[8] Pavlović, D. and Escardó, M. H. 1998. Calculus in coinductive form, *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science,* 408–417. IEEE Computer Science Society Press.

[9] Peterson, J. and Hammond, K. (editors) 1997. *Report on the Programming Language Haskell.* Available from http://www.haskell.org.

[10] Rutten, J. J. M. M. 1999. Automata, power series, and coinduction: taking input derivatives seriously. *Proceedings of ICALP ’99,* J. Wiedermann, P. van Emde Boas and M. Nelson, editors, 645–654. Springer LNCS 1644.