



# Algebraic Specification and Coalgebraic Synthesis of Mealy Automata

J.J.M.M. Rutten<sup>1</sup>

*CWI and VUA  
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

---

## Abstract

We introduce the notion of functional stream derivative, generalising the notion of input derivative of rational expressions (Brzowski 1964) to the case of stream functions over arbitrary input and output alphabets. We show how to construct Mealy automata from algebraically specified stream functions by the symbolic computation of functional stream derivatives. We illustrate this construction in full detail for various bitstream functions specified in the algebraic calculus of the 2-adic numbers. This work is part of a larger ongoing effort to specify and model component connector circuits in terms of (functions and relations on) streams.

*Keywords:* Stream function, functional stream derivative, Mealy automaton, bitstream, binary arithmetic, 2-adic integer.

---

## 1 Introduction

In [7], Brzowski showed how to construct a deterministic finite automaton for a rational expression by computing its finitely many *derivatives*, herewith lifting the well-known fact that rational languages have a finite number of (left) quotients, to the symbolic level of expressions. Since then, various applications and generalisations have been studied. In [1], Antimirov introduced the notion of *partial derivative* and used it to construct nondeterministic finite automata. In [19,20], we reformulated Brzowski's original approach in coalgebraic terms and generalised it to formal power series over arbitrary semirings, providing at the same time a generalisation of Antimirov's results. A similar generalisation to formal power series and rational expressions with multiplicities was found, independently, by Lombardy and Sakarovitch [15,16].

Here we present yet another generalisation of Brzowski's construction. We look at deterministic *Mealy automata* [9], with inputs and outputs over arbitrary alpha-

---

<sup>1</sup> Email: [janr@cwi.nl](mailto:janr@cwi.nl)

bets (cf. Example 2.4). We use *stream functions* from infinite sequences of inputs to infinite sequences of outputs to specify their behaviour. We introduce the (semantic) notion of *functional stream derivative* (Section 2), which can be used to construct for any stream function a minimal Mealy automaton. Functional stream derivatives already occur in [17] under the name of *state* (of a sequential function). Also minimisation of Mealy automata is well known [9]. The main contribution of the present paper is the insight that functional stream derivatives can often be computed *symbolically* from algebraic expressions that specify stream functions. In this manner, a Mealy automaton can be constructed from an algebraic expression by symbolic manipulation. We describe this construction in full detail for various functions on *bitstreams* (infinite sequences of 0's and 1's). We specify bitstream functions in the algebra of the *2-adic numbers*, and construct Mealy automata that implement these functions by symbolically computing their functional stream derivatives (Sections 3 through 5). In general, this yields infinite automata, but we will also characterise some families of algebraic expressions, for which the construction yields a finite (and minimal) automaton.

Contributions and related work will be discussed in Section 6. This will include a discussion of the relevance of our construction for the design of digital circuits. Here we want to point out that the present work is part of a broader ongoing research effort to develop models and specification formalisms for component connector circuits. In [5] and [4], we used relations on streams and so-called constraint automata as models of Arbab's [2] component connector calculus Reo. There we showed how to go from connector circuits to (relations on) streams and also from connector circuits to automata. In addition, initial ideas how to construct circuits from automata (which amounts to a non-trivial generalisation of well-known techniques from logic design) are described in [3]. We intend to generalise the techniques of the present paper to the specification and symbolic construction of (constraint) automata for component connector circuits.

## 2 Mealy automata

We give the basic definitions on Mealy automata and streams. We introduce the notion of functional stream derivative and show how it can be used to characterise minimal automata.

Let  $A$  and  $B$  be arbitrary sets. A *Mealy automaton*  $(S, \phi)$  with inputs in  $A$  and outputs in  $B$  consists of a set of states  $S$  and a transition function  $\phi : S \rightarrow (B \times S)^A$ . This function maps a state  $s_0 \in S$  to a function  $\phi(s_0) : A \rightarrow (B \times S)$ , which produces for every input  $a \in A$  a unique pair  $\langle b, s_1 \rangle$ , consisting of the output  $b$  and the next state  $s_1$ .

(Mealy automata are also called *sequential machines* [9]. Generalisations include transition functions that are partial and that may map into  $B^* \times S$  instead of  $B \times S$ . In more recent references, the latter are called (sub)sequential transducers [18,8]. In coalgebraic terms, a Mealy automaton is a coalgebra of the functor  $F : \text{Set} \rightarrow \text{Set}$  on the category of sets and functions, which is defined for any set  $S$  by  $F(S) =$

$(B \times S)^A$ .)

We shall use the following notation:

$$s_0 \xrightarrow{a|b} s_1 \iff \phi(s_0)(a) = \langle b, s_1 \rangle$$

We call a Mealy automaton *binary* if inputs and outputs are taken from the set  $2 = \{0, 1\}$ .

We define the set  $A^\omega$  of *streams* over an arbitrary set  $A$  by  $A^\omega = \{\sigma \mid \sigma : \mathbb{N} \rightarrow A\}$ . Elements  $\sigma \in A^\omega$  will be denoted by  $\sigma = (\sigma(0), \sigma(1), \sigma(2), \dots)$ . We define the *stream derivative* of a stream  $\sigma$  by

$$\sigma' = (\sigma(1), \sigma(2), \sigma(3), \dots)$$

and we call  $\sigma(0)$  the *initial value* of  $\sigma$ . We shall use the following notation, for  $a_0, \dots, a_n \in A$  and  $\sigma \in A^\omega$ :

$$a_0 : \dots : a_n : \sigma = (a_0, \dots, a_n, \sigma(0), \sigma(1), \sigma(2), \dots)$$

For instance, we have  $\sigma = \sigma(0) : \sigma'$ . We call a function  $f : A^\omega \rightarrow B^\omega$  *causal* if for any  $\sigma \in A^\omega$ , the  $n$ -th element of the stream  $f(\sigma)$  depends only on the first  $n$  elements of  $\sigma$ . More formally,  $f$  is causal if

$$f(a_0 : \dots : a_n : \sigma)(n) = f(a_0 : \dots : a_n : \tau)(n)$$

for all  $n \geq 0$ ,  $a_0, \dots, a_n \in A$ ,  $\sigma, \tau \in A^\omega$ . As we shall see shortly, causal stream functions typically arise as descriptions of the behaviour of Mealy automata. Note that the composition of causal functions is again causal. (Causal functions are sometimes also called synchronous or letter-to-letter.)

The following elementary definition introduces the notion of functional stream derivative, on which later a new symbolic algorithm for the construction of Mealy automata will be based.

**Definition 2.1** [functional stream derivative] For a causal function  $f : A^\omega \rightarrow B^\omega$  and  $a \in A$ , we define the *initial output* of  $f$  (on input  $a$ ) by  $f[a] = f(a : \sigma)(0)$ , where  $\sigma \in A^\omega$  is arbitrary. We define the *functional stream derivative* of  $f$  (on input  $a$ ) as the function  $f_a : A^\omega \rightarrow B^\omega$ , defined for all  $\sigma \in A^\omega$ , by  $f_a(\sigma) = f(a : \sigma)'$ .  $\square$

Note that in the definition of  $f[a]$ , the actual value of  $\sigma$  is irrelevant because  $f$  is causal. Our notation  $f[a]$  should not be read as:  $f$  applied to the argument  $a$  (which does not make sense as  $f$  takes infinite streams as arguments), but just as a shorthand for  $f(a : \sigma)(0)$ . One could say that on a stream (of inputs)  $\sigma$ , the functional stream derivative  $f_a$  acts as  $f$  would “after it had seen  $a$  first”.

Let  $\Gamma = \{f \mid f : A^\omega \rightarrow B^\omega \mid f \text{ is causal}\}$ . We define a function  $\pi : \Gamma \rightarrow (B \times \Gamma)^A$ , for  $f \in \Gamma$  and  $a \in A$ , by  $\pi(f)(a) = \langle f[a], f_a \rangle$  (note that  $f_a$  is causal when  $f$  is). This gives us an (infinite) Mealy automaton  $(\Gamma, \pi)$  with transitions

$$f \xrightarrow{a|f[a]} f_a$$

Next we characterise  $(\Gamma, \pi)$  using the following notion. A *homomorphism* of automata  $(S, \phi : S \rightarrow (B \times S)^A)$  and  $(T, \psi : T \rightarrow (B \times T)^A)$  is a function  $h : S \rightarrow T$  that preserves transitions: if  $\phi(s_0)(a) = \langle b, s_1 \rangle$  then  $\psi(h(s_0))(a) = \langle b, h(s_1) \rangle$ ; in other words,

$$s_0 \xrightarrow{a|b} s_1 \Rightarrow h(s_0) \xrightarrow{a|b} h(s_1)$$

**Proposition 2.2**  $(\Gamma, \pi)$  is a final Mealy automaton: for every Mealy automaton  $(S, \phi)$  (with inputs in  $A$  and outputs in  $B$ ), there exists a unique homomorphism  $h : (S, \phi) \rightarrow (\Gamma, \pi)$ .

**Proof.** For a Mealy automaton  $(S, \phi)$ , we define a function  $h : S \rightarrow \Gamma$ . For  $s_0 \in S$ , we define a function  $h(s_0) : A^\omega \rightarrow B^\omega$  by considering, for  $\sigma \in A^\omega$  and  $k \geq 0$ , the (unique) corresponding sequence of transitions

$$s_0 \xrightarrow{\sigma(0)|b_0} s_1 \xrightarrow{\sigma(1)|b_1} \dots \xrightarrow{\sigma(k)|b_k} s_{k+1}$$

and putting  $h(s_0)(\sigma)(k) = b_k$ . It is not very difficult to verify that  $h(s_0)$  is causal, and that the function  $h$  defined in this way is a homomorphism, which is moreover unique. □

We call the stream function  $h(s_0)$  above the (input-output) *behaviour* of  $s_0$ . For a causal function  $f \in \Gamma$ , we say that a state  $s_0$  in an automaton  $S$  *implements*  $f$  if  $f = h(s_0)$ .

In algebra, the behaviour of Mealy automata is typically described in terms of functions of type  $A^+ \rightarrow B$ . Although this set is isomorphic to the final coalgebra  $\Gamma$ , we prefer to work with the latter because of its, as we shall see later, richer algebraic structure.

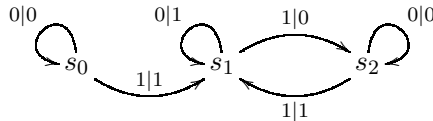
The universality of  $\Gamma$  can be expressed in yet another way. We need the following notion. For a state  $s_0 \in S$  of a Mealy automaton  $(S, \phi)$ , let  $\langle s_0 \rangle \subseteq S$  denote the smallest subset that contains  $s_0$  and is closed under transitions (for any inputs). Clearly,  $\langle s_0 \rangle$  is also a *subautomaton* of  $(S, \phi)$ , by taking as its transition function the restriction of  $\phi$  to the set  $\langle s_0 \rangle$ . We call  $\langle s_0 \rangle$  the subautomaton of  $(S, \phi)$  *generated* by  $s_0$ .

**Corollary 2.3** (a) For a causal function  $f \in \Gamma$ , the (state  $f$  in the) subautomaton  $\langle f \rangle \subseteq \Gamma$  implements  $f$ . (b) Moreover,  $\langle f \rangle$  is the minimal Mealy automaton (having the smallest number of states) that implements  $f$ .

**Proof.** Statement (a) follows from the fact that the identity function  $id : \Gamma \rightarrow \Gamma$  is a homomorphism. For (b), let  $(S, \phi)$  be a Mealy automaton and consider a state  $s_0 \in S$  that implements  $f$ , that is:  $h(s_0) = f$ , with  $h$  as in Proposition 2.2. Without loss of generality, we may assume that all states in  $S$  are reachable from  $s_0$ , that is,  $\langle s_0 \rangle = S$ . (If not, we simply take  $\langle s_0 \rangle$  instead of  $S$  to begin with.) Because  $h$  is a homomorphism (and thus preserves transitions), it follows that the image  $h(S)$  of  $S$  under  $h$  is a subautomaton of  $\Gamma$ ; moreover,  $h(S) = h(\langle s_0 \rangle) = \langle h(s_0) \rangle = \langle f \rangle$ . Thus the size of  $S$  is at least the size of  $\langle f \rangle$ . □

According to the definition of  $(\Gamma, \pi)$ , the minimal Mealy automaton  $\langle f \rangle$  that implements  $f \in \Gamma$  can be constructed by computing repeatedly all functional stream derivatives of  $f$ . In Section 5, we shall describe how to construct binary automata from algebraically specified stream functions by symbolic computation.

**Example 2.4** We illustrate the notions and results above with a simple example. Consider a binary Mealy automaton with state space  $S = \{s_0, s_1, s_2\}$  and transition function  $f : S \rightarrow (2 \times S)^2$  defined by the following picture:



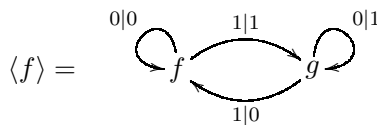
It is a single bit binary counter. Starting in state  $s_0$ , this automaton outputs 0 when an even number of 1’s has been input and it outputs 1 when an odd number of 1’s has been input. Note that we have introduced, on purpose, some redundancy: states  $s_0$  and  $s_2$  are equivalent and can be identified (as we shall see below). Next consider functions  $f, g : 2^\omega \rightarrow 2^\omega$  defined, for  $\sigma \in 2^\omega, k \geq 0$ , by

$$f(\sigma)(k) = \begin{cases} 0 & \text{if } (\sigma(0), \dots, \sigma(k)) \text{ contains an even number of 1's} \\ 1 & \text{otherwise} \end{cases}$$

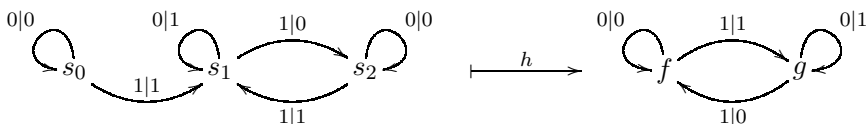
and  $g(\sigma)(k) = 1 - f(\sigma)(k)$ . We have the following initial outputs and functional stream derivatives:

$$f[0] = 0, \quad f[1] = 1, \quad g[0] = 1, \quad g[1] = 0, \quad f_0 = f, \quad f_1 = g, \quad g_0 = g, \quad g_1 = f$$

It follows that a minimal Mealy automaton that implements  $f$  is given by



Also note that with  $h$  as in Proposition 2.2, we have  $h(s_0) = h(s_2) = f$  and  $h(s_1) = g$ , and



Thus  $h$  maps  $S = \langle s_0 \rangle$  to its minimization  $\langle f \rangle \subseteq \Gamma$ , on the right. □

As we mentioned in the introduction, the notion of functional stream derivative already occurs in [17], and is called *state* there. It is also a variation on the classical notion of derivative (or inverse) of functions from  $A^*$  to  $B^*$  [9]. Also Proposition

2.2 and Corollary 2.3 are essentially reformulations of classical results. The contribution of the present paper consists of the observation that functional stream derivatives can be *symbolically* computed from algebraic expressions that specify stream functions. This will be carried out for certain functions on bitstreams, in Section 5, using the algebra of the 2-adic numbers that we introduce next.

### 3 The 2-adic operators

In Section 5, we shall construct minimal Mealy automata from stream functions by computing functional stream derivatives. Although this method works for automata and functions on streams over arbitrary alphabets, we shall illustrate it for the case of binary automata and bitstream functions. More specifically still, we shall look at bitstream functions that are specified in terms of a number of basic operators on bitstreams, the so-called 2-adic operators, which we introduce in the present section. (Other types of operators will be mentioned in the conclusions.)

Let  $2^\omega$  be the set of bitstreams: infinite sequences of 0's and 1's. Before introducing the 2-adic operators, we first recall (cf. [13,10]) that bitstreams are also known as *2-adic numbers*, since they can be viewed as the binary representations of ordinary numbers. For rational numbers with odd denominator, this works as follows. We define the *binary representation*  $B(q)$  of a rational  $q \in \hat{Q} = \{n/(2m+1) \mid n, m \in \mathbb{Z}\}$  as the unique stream satisfying the following system of equations (one for each  $q$ ):

$$B(q)' = B((q - \text{odd}(q))/2), \quad B(q)(0) = \text{odd}(q)$$

where  $\text{odd}(n/2m+1) = 1$ , if  $n$  is odd, and  $= 0$  if  $n$  is even. These equations define streams  $B(q)$  by specifying their stream derivative  $B(q)'$  and initial value  $B(q)(0)$ . Therefore they are called *stream differential equations* (see [20] for an overview). It is not very difficult to see that the above equations define an inclusion  $B : \hat{Q} \rightarrow 2^\omega$ . (The definition does not work for rationals with even denominator.) Here are some examples; note that the least significant bit is always on the left:  $B(13) = (1, 0, 1, 1, 0, 0, 0, \dots)$ ,  $B(-5) = (1, 1, 0, 1, 1, 1, \dots)$ , and  $B(1/5) = (1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, \dots)$ . In general, the bitstream of a positive integer ends with 0's, the bitstream of a negative integer ends with 1's, and the bitstream of a rational is eventually periodic. (All of this is well known; see the aforementioned references.)

Next we introduce the *2-adic operators*, with which one can calculate with bitstreams in the same manner as with ordinary numbers. We need the following preliminaries. The Boolean operators on  $2 = \{0, 1\}$  are defined, for all  $a, b \in 2$ , as usual:  $a \vee b = \max\{a, b\}$ ,  $a \wedge b = \min\{a, b\}$ ,  $\neg a = 1 - a$ , and  $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$  (exclusive or). Classically, the 2-adic operators are defined in terms of formal power series (see the aforementioned references). Here we take the 2-adic operators simply as functions from bitstreams to bitstreams, and we define them by means of stream differential equations. These definitions are equivalent to the classical ones, but have the advantage that with their help, it will be immediate to compute functional stream derivatives, which will be used in the construction of binary automata (Sec-

tion 5). For bitstreams  $\sigma, \tau \in 2^\omega$ , we define  $\sigma + \tau$ ,  $-\sigma$ ,  $\sigma \times \tau$ , and  $1/\sigma$  as the unique streams satisfying the system of equations below (we use the following notation: for  $a \in 2$ , we write  $[a] = (a, 0, 0, 0, \dots)$ ):

derivative:	initial value:	name:
$(\sigma + \tau)' = (\sigma' + \tau') + [\sigma(0) \wedge \tau(0)]$	$(\sigma + \tau)(0) = \sigma(0) \oplus \tau(0)$	sum
$(-\sigma)' = -(\sigma' + [\sigma(0)])$	$(-\sigma)(0) = \sigma(0)$	minus
$(\sigma \times \tau)' = (\sigma' \times \tau) + ([\sigma(0)] \times \tau')$	$(\sigma \times \tau)(0) = \sigma(0) \wedge \tau(0)$	product
$(1/\sigma)' = -(\sigma' \times (1/\sigma))$	$(1/\sigma)(0) = 1$	inverse

As usual, we shall write  $\tau/\sigma$  for  $\tau \times (1/\sigma)$ ; note that  $1/\sigma$  is defined only for  $\sigma$  with  $\sigma(0) = 1$ . It is not difficult to prove that this system uniquely determines these four operators (see [21] for details), and that they are causal.

We briefly explain the form of these equations. The sum  $\sigma + \tau$  is computed by elementwise addition but with the proviso that ‘overflow’ bits are carried over to higher-order positions (that is to say, next right in the stream). This explains the initial value  $(\sigma + \tau)(0) = \sigma(0) \oplus \tau(0)$  and the presence of the ‘carry’ term  $[\sigma(0) \wedge \tau(0)]$  in the derivative  $(\sigma + \tau)'$ . The definition of  $-\sigma$  (aka two’s complement) is completely determined by the ‘requirement’ that  $\sigma + (-\sigma) = [0]$ : taking initial values on both sides implies that  $\sigma(0) \oplus ((-\sigma)(0)) = 0$ , whence  $(-\sigma)(0) = \sigma(0)$ ; and taking derivatives on both sides implies  $\sigma' + (-\sigma)' + [\sigma(0)] = [0]$ , from which the shape of the differential equation for  $-\sigma$  follows. Multiplication is defined in terms of shift and addition. The definition of inverse, finally, follows from the ‘requirement’ that  $\sigma \times (1/\sigma) = [1]$ , for any  $\sigma \in 2^\omega$  with  $\sigma(0) = 1$ , again by taking initial values and derivatives on both sides of that equation.

The reader familiar with the definition of input derivatives of rational expressions [7] will see certain differences. For instance, we have only one type of derivative here, whereas for rational expressions  $E$  over an alphabet  $A$  we have one input derivative  $E_a$  for every  $a \in A$ . (As we have seen in Section 2, derivatives with respect to input *do* play a role in the definition of *functional* stream derivative.) But there are also similarities. For instance, the derivative of the product above is very similar to the input derivative of the concatenation product of rational expressions:  $(E \times F)_a = (E_a \times F) + (\sigma(E) \times F_a)$ . For an explanation of the common basis of both types of derivatives, we refer to [20].

## 4 A bit of 2-adic calculus

We briefly review some basic properties of the 2-adic operators that will be useful for the computations in Section 5. It is well known that  $2^\omega$  with the 2-adic operators, is a commutative ring (and an integral domain). So we have the usual commutativity, associativity, and distributivity laws for sum and product. So far we have constants  $[0] = (0, 0, 0, \dots)$  and  $[1] = (1, 0, 0, 0, \dots)$ . Since  $(1, 0, 0, 0, \dots) + (1, 1, 1, \dots) = (0, 0, 0, \dots)$ , we have  $-[1] = (1, 1, 1, \dots)$ . We shall often simply write 0, 1, and  $-1$

for  $[0]$ ,  $[1]$ , and  $-[1]$ , respectively; the context should make clear whether we are talking about streams or numbers. In order to formulate some further properties, we introduce yet another constant:  $X = (0, 1, 0, 0, 0, \dots)$ , which plays the role of formal variable (as in formal power series). Here is a number of useful identities. We have omitted their proofs, which are straightforward (see also [21]).

**Lemma 4.1** *We have derivatives:  $1' = 0' = 0$ ,  $(-1)' = -1$ ,  $X' = 1$ , and initial values:  $0(0) = X(0) = 0$ ,  $1(0) = -1(0) = 1$ . For  $\sigma, \tau \in 2^\omega$  and  $n \geq 0$ :  $X \times \sigma = (0, \sigma(0), \sigma(1), \sigma(2), \dots)$ ,  $(X \times \sigma)' = \sigma$ ,  $(-X \times \sigma)' = -\sigma$ ,  $(-\sigma)' = \sigma' - \sigma$ ,  $\sigma + \sigma = X \times \sigma$ , and*

$$(\sigma \times \tau)' = \begin{cases} \sigma' \times \tau & \text{if } \sigma(0) = 0 \\ (\sigma' \times \tau) + \tau' & \text{if } \sigma(0) = 1 \end{cases} \tag{1}$$

$$(\sigma/\tau)' = \begin{cases} \sigma'/\tau & \text{if } \sigma(0) = 0 \text{ and } \tau(0) = 1 \\ (\sigma' - \tau')/\tau & \text{if } \sigma(0) = 1 \text{ and } \tau(0) = 1 \end{cases} \tag{2}$$

Immediate corollaries are  $X^2 = (0, 0, 1, 0, 0, 0, \dots)$ ,  $X^3 = (0, 0, 0, 1, 0, 0, 0, \dots)$ , etc. Particularly lovely are identities such as  $1+1 = X$ ,  $X^n + X^n = X^{n+1}$  and  $(X^{n+1})' = X^n$ . Using the constant  $X$ , we define the following notions.

**Definition 4.2** We call a stream  $\pi = c_0 + c_1X + c_2X^2 + \dots + c_kX^k$ , where all  $c_i$  are either 0, 1, or  $-1$ , a *polynomial*. We define  $deg(\pi) = k$  if  $c_k \neq 0$ . We call  $\sigma \in 2^\omega$  *rational* if  $\sigma = \pi/\rho$ , for polynomials  $\pi$  and  $\rho$  with  $\rho(0) = 1$ . □

Note that in the definition above, negative coefficients are allowed. In particular, also  $-1 = (1, 1, 1, \dots)$  is considered a polynomial. For an example of rational stream, recall the function  $B : \hat{Q} \rightarrow 2^\omega$  from Section 3 that assigns to a rational number (with odd denominator) its binary representation. Because  $B(1) = [1]$ ,  $B(2) = X$ , and  $B$  commutes with the operators of sum, minus, product, and inverse (in other words,  $B$  is a homomorphism of integral domains), it follows that  $B(q)$  is a rational stream, for all  $q \in \hat{Q}$ .

For  $\sigma \in 2^\omega$  and  $n \geq 0$ , we define the  $n$ -th derivative  $\sigma^{(n)}$  of  $\sigma$  by  $\sigma^{(0)} = \sigma$  and  $\sigma^{(n+1)} = (\sigma^{(n)})'$ . The following proposition is a variation on the observation by Brzozowski [7] that rational expressions have only finitely many distinct input derivatives.

**Proposition 4.3** *A rational stream has only finitely many distinct stream derivatives.*

**Proof.** Consider polynomials  $\pi, \rho$  (with  $\rho(0) = 1$ ). By identity (2) and because  $deg(\pi' - \rho') \leq M = \max\{deg(\pi), deg(\rho)\}$ , it follows for all  $n \geq 0$  that the  $n$ -th derivative  $(\pi/\rho)^{(n)}$  is of the form  $\kappa/\rho$  for some polynomial  $\kappa$  of degree  $deg(\kappa) \leq M$ . As there are only finitely many different such polynomials, the proposition follows. □

This proposition is equivalent to the well known fact, mentioned earlier, that binary representations of rational numbers (with odd denominator) are eventually

periodic. The formulation of Proposition 4.3 in terms of stream derivatives will be particularly useful in Section 5. There it will be used to show that certain stream functions have only finitely many functional stream derivatives and, consequently, give rise to a finite (and minimal) Mealy automaton. In order to obtain some experience with the computation of stream derivatives, we conclude this section with the calculation of the canonical forms of the binary representations  $B(q)$  of rational numbers (with odd denominator)  $q$ . We shall use the following notation, for  $a_i, b_j \in 2$ :

$$a_0 \cdots a_k (b_0 \cdots b_l) = (a_0, \dots, a_k, b_0, \dots, b_l, b_0, \dots, b_l, \dots)$$

As an example, we compute the binary representations of 5,  $-5$ , and  $1/5$ . For integers, things are very simple:

$$B(5) = B(1 + 2^2) = B(1) + B(2)^2 = 1 + X^2 = 101(0)$$

$$B(-5) = -B(5) = -1 - X^2 = 1 + X - X^3 = 110(1)$$

where we used  $-X^k = X^k - X^{k+1}$ , all  $k \geq 0$ . For  $B(1/5) = 1/B(5) = 1/1 + X^2$ , we compute the repeated derivatives of  $1/1 + X^2$ , using the defining differential equations from Section 3 and some of the identities of Lemma 4.1. In particular, we shall use below that

$$(1 + X^2)' = (1)' + (X^2)' + [(1)(0) \wedge (X^2)(0)] = 0 + X + 0 = X$$

$$(-1 - X)' = (-1)' + (-X)' + [(-1)(0) \wedge (-X)(0)] = (-1) + (-1) + 0 = -X$$

The repeated derivatives of  $1/1 + X^2$  are as follows:

$$\begin{aligned} (1/1 + X^2)' &= (1' - (1 + X^2)')/1 + X^2 = -X/1 + X^2 \\ (-X/1 + X^2)' &= -1/1 + X^2 \\ (-1/1 + X^2)' &= ((-1)' - (1 + X^2)')/1 + X^2 = (-1 - X)/1 + X^2 \\ ((-1 - X)/1 + X^2)' &= ((-1 - X)' - (1 + X^2)')/1 + X^2 \\ &= ((-X) - (X))/1 + X^2 = -X^2/1 + X^2 \\ (-X^2/1 + X^2)' &= -X/1 + X^2 \end{aligned}$$

These are all the different derivatives of  $1/1 + X^2$ . Taking their initial values, and using the fact that  $\sigma = \sigma(0) : \sigma'$ , for any  $\sigma \in 2^\omega$ , we find  $B(1/5) = 1(0110)$ .

## 5 Synthesis of binary automata

In Definition 2.1, we introduced the new notion of functional stream derivative. The present section contains our main contribution: we show how to use functional stream derivatives to construct minimal Mealy automata. We shall present a number of examples and one, more systematic but fairly modest, general result. (All of this will be about bitstream functions and binary Mealy automata, but we repeat

that our method is more general than that; other cases will be discussed in Section 6.) We shall use the following identities: for all  $\sigma \in 2^\omega$ ,

$$0 : \sigma = X \times \sigma, \quad 1 : \sigma = 1 + (X \times \sigma)$$

Recall from Definition 2.1 the definition of the functional stream derivatives of a causal function  $f : 2^\omega \rightarrow 2^\omega$ , for all  $\sigma \in 2^\omega$ :

$$f_0(\sigma) = f(0 : \sigma)', \quad f_1(\sigma) = f(1 : \sigma)',$$

Using the two identities above, we have the following formulae: for all  $\sigma \in 2^\omega$ ,

$$f_0(\sigma) = f(X \times \sigma)', \quad f_1(\sigma) = f(1 + (X \times \sigma))'$$

Also, we shall use the following notation for repeated functional stream derivatives: for  $w \in 2^*$  and  $a \in 2$ , we define  $f_\varepsilon = f$  (where  $\varepsilon$  is the empty word) and  $f_{wa} = (f_w)_a$ . Here is a first example of our construction. Consider the following function  $f : 2^\omega \rightarrow 2^\omega$ :

$$f(\sigma) = (1 + X) \times \sigma$$

(Note that in this expression  $X$  denotes the constant stream  $(0, 1, 0, 0, 0, \dots)$  and  $\sigma$  is the function variable.) In the terminology of Section 3, this function multiplies a 2-adic number  $\sigma \in 2^\omega$  by  $1 + X = B(3)$ , the binary representation of the natural number 3. Note that  $f$  is causal and let  $\Gamma$  as before denote the set of all causal functions from  $2^\omega$  to  $2^\omega$ . Recall (Corollary 2.3) that  $\langle f \rangle \subseteq \Gamma$ , the subautomaton of  $\Gamma$  generated by  $f$ , is the smallest Mealy automaton implementing  $f$ . We construct  $\langle f \rangle$  by computing the repeated functional stream derivatives of  $f$  (using the defining differential equations from Section 3 and some of the identities from Section 4):

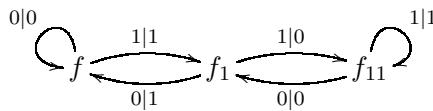
$$\begin{aligned} f_0(\sigma) &= f(X \times \sigma)' = ((1 + X) \times (X \times \sigma))' \\ &= ((1 + X)' \times (X \times \sigma)) + (X \times \sigma)' \quad [\text{note that } (1 + X)(0) = 1] \\ &= (X \times \sigma) + \sigma = f(\sigma) \\ f_1(\sigma) &= f(1 + (X \times \sigma))' = (((1 + X) \times (1 + (X \times \sigma))))' \\ &= ((1 + X)' \times (1 + (X \times \sigma))) + (1 + (X \times \sigma))' \\ &= 1 + (X \times \sigma) + \sigma = f(\sigma) + 1 \\ f_{10}(\sigma) &= f_1(X \times \sigma)' = (((1 + X) \times (X \times \sigma)) + 1)' \\ &= ((1 + X) \times (X \times \sigma))' + 1' \\ &\quad [\text{since } ((1 + X) \times (X \times \sigma))(0) \wedge 1(0) = 0] \\ &= f(\sigma) \\ f_{11}(\sigma) &= f_1(1 + (X \times \sigma))' = (((1 + X) \times (1 + (X \times \sigma))))' + 1' \\ &= ((1 + X) \times (1 + (X \times \sigma)))' + 1' + 1 \\ &= f_1(\sigma) + 1 = f(\sigma) + 1 + 1 = f(\sigma) + X \\ f_{110}(\sigma) &= f_{11}(X \times \sigma)' = (f(X \times \sigma) + X)' = f(X \times \sigma)' + 1 = f_1(\sigma) \\ f_{111}(\sigma) &= f_{11}(1 + (X \times \sigma))' = (f(1 + (X \times \sigma)) + X)' \\ &= f(1 + (X \times \sigma))' + 1 = f_{11}(\sigma) \end{aligned}$$

Computing further derivatives will not yield any new functions, so the automaton  $\langle f \rangle$  consists of the following three states:  $\{f, f_1, f_{11}\}$ . Transitions are given by initial outputs, which are computed as follows (below  $\sigma$  is an arbitrary stream):

$$f[0] = f(0 : \sigma)(0) = f(X \times \sigma)(0) = ((1 + X) \times (X \times \sigma))(0) = 0$$

$$f[1] = f(1 : \sigma)(0) = f(1 + (X \times \sigma))(0) = ((1 + X) \times (1 + (X \times \sigma)))(0) = 1$$

Similarly, we compute  $f_1[0] = 1, f_1[1] = 0, f_{11}[0] = 0,$  and  $f_{11}[1] = 1$ . And so we find the following minimal automaton for  $f$ :



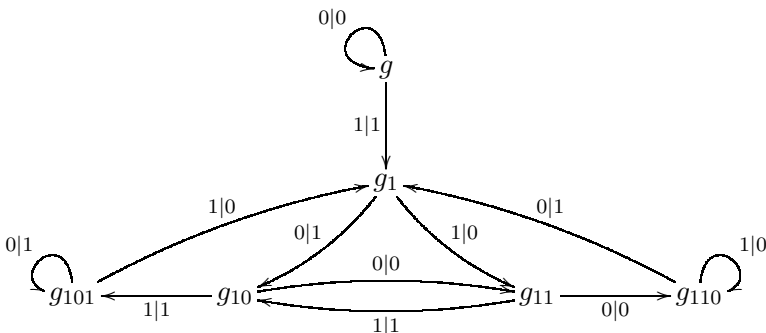
For a second example, consider the function  $g : 2^\omega \rightarrow 2^\omega$  defined, for  $\sigma \in 2^\omega$ , by

$$g(\sigma) = \sigma \times (1/1 - X - X^2)$$

It multiplies a 2-adic number  $\sigma$  by  $-1/5$ , since  $B(-1/5) = 1/1 - X - X^2$ . Computing the functional stream derivatives of  $g$  gives the following distinct functions (omitting details):

$$\begin{aligned} g_1(\sigma) &= g(\sigma) + ((1 + X)/1 - X - X^2) \\ g_{10}(\sigma) &= g(\sigma) + (X^2/1 - X - X^2) \\ g_{11}(\sigma) &= g(\sigma) + (X/1 - X - X^2) \\ g_{101}(\sigma) &= g(\sigma) + ((1 + X^2)/1 - X - X^2) \\ g_{110}(\sigma) &= g(\sigma) + (1/1 - X - X^2) \end{aligned}$$

After computing the corresponding initial outputs, one finds the following minimal Mealy automaton implementing  $g$ :



Our third and last example illustrates that our method also works for multiplication with an arbitrary rational number. We consider the function  $h : 2^\omega \rightarrow 2^\omega$  defined, for  $\sigma \in 2^\omega$ , by

$$h(\sigma) = \sigma \times (1 + X/1 - X - X^2)$$

which multiplies a 2-adic number  $\sigma$  by  $-3/5$ . We do not mention any details but only present the state space of the resulting automaton, which as before is obtained by computing repeatedly the functional stream derivatives of  $h$ . It consists of all expressions

$$h(\sigma) + (-E/1 - X - X^2),$$

with  $E \in \{0, 1, X, 1 + X, X^2, 1 + X^2, X + X^2, 1 + X + X^2\}$

Thus an automaton is obtained with 8 states.

The above procedure always yields a finite automaton for functions of the following form.

**Theorem 5.1** *A function  $h : 2^\omega \rightarrow 2^\omega$  of the form  $h(\sigma) = \sigma \times \phi$ , for all  $\sigma \in 2^\omega$  and for fixed rational stream  $\phi$ , has only finitely many distinct functional stream derivatives. As a consequence,  $\langle h \rangle$  is a minimal finite automaton implementing  $h$ .*

**Proof.** This is an immediate consequence of Proposition 4.3: rational streams have only finitely many distinct stream derivatives. Let  $\phi = \alpha/\beta$ , for polynomial streams  $\alpha, \beta$ . One can show that any repeated functional stream derivative of  $h$  is always of the form  $h(\sigma) + (\gamma/\beta)$ , for polynomials  $\gamma$  with degree  $\text{deg}(\gamma) \leq \text{deg}(\alpha) + \text{deg}(\beta)$ . There are only finitely many such functions. □

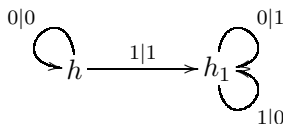
There are various easy generalisations of this theorem that are omitted here. For instance, it also holds for functions of the form  $h(\sigma) = (\sigma \times \phi) + \psi$  for fixed rational streams  $\phi$  and  $\psi$ . Also, it is straightforward to formulate a similar theorem for functions with many inputs and many outputs.

(See Section 6 for a discussion of ongoing work, by David de Oliveira Costa and Helle Hvid Hansen, on an implementation of the algorithm underlying Theorem 5.1. For instance, our last example above was computer generated. Interestingly, experimental data have by now lead to a number of conjectures about both size and structure of the state space of the resulting automata.)

In conclusion of this section, we look at two examples that illustrate that our construction is not limited to functions of the form of Theorem 5.1. Let  $h(\sigma) = -\sigma$ , for all  $\sigma \in 2^\omega$ . Computing derivatives gives

$$\begin{aligned} h_0(\sigma) &= h(X \times \sigma)' = -(X \times \sigma)' = -\sigma = h(\sigma) \\ h_1(\sigma) &= h(1 + (X \times \sigma))' = -(1 + (X \times \sigma))' \\ &= -((1 + (X \times \sigma))' + 1) = -(\sigma + 1) \\ h_{10}(\sigma) &= -((X \times \sigma) + 1)' = -(((X \times \sigma) + 1)' + 1) \\ &= -(\sigma + 1) = h_1(\sigma) \\ h_{11}(\sigma) &= -(1 + (X \times \sigma) + 1)' \\ &= -(((X \times \sigma) + X))' = -(\sigma + 1) = h_1(\sigma) \end{aligned}$$

Computing initial outputs then gives the following minimal implementation of the 2-adic minus:



Our next and last example shows that not all is finite, in automaton land. Let  $i : 2^\omega \rightarrow 2^\omega$  be defined, for all  $\sigma \in 2^\omega$ , by  $i(\sigma) = 1/(1 - (X \times \sigma))$ . Computing functional stream derivatives gives

$$\begin{aligned} i_0(\sigma) &= (1/(1 - (X \times (X \times \sigma))))' = (X \times \sigma)/(1 - (X^2 \times \sigma)) \\ i_{00}(\sigma) &= ((X \times (X \times \sigma))/(1 - (X^2 \times (X \times \sigma))))' \\ &= (X \times \sigma)/(1 - (X^3 \times \sigma)) \end{aligned}$$

By induction, one has  $i_{0^n}(\sigma) = (X \times \sigma)/(1 - (X^{n+1} \times \sigma))$ , for all  $n \geq 1$ . Since all these functions are different (just take  $\sigma = 1$  for an argument), and since  $\langle i \rangle$  is the minimal automaton implementing  $i$ , it follows that there is *no* finite automaton implementing the operation of 2-adic division. Although some negative results of this type already exist, the last example above offers a new and very general way of proving them.

## 6 Discussion and related work

*Contributions:* (a) the observation that the set of all causal stream functions is a final Mealy automaton, of which the coalgebra structure is given by the notion of (initial value and) functional stream derivative; (b) a new algorithm for the symbolic construction of minimal Mealy automata from algebraically specified causal functions, in particular 2-adic arithmetical bitstream functions.

*Other binary algebras:* In [21], we study a number of other algebraic structures on  $2^\omega$ , every time defining the operators of the algebra by means of stream differential equations. As a consequence, the construction of Mealy automata by means of functional stream derivatives, is in principle applicable to functions specified in each of those algebras. More interesting still, the same holds for functions that are specified by so-called *mixed* expressions, in which operators from different algebras are used together. We lack the space here to include further examples. In general, the resulting automata will be infinite, but for certain classes of functions it can be proven that a finite automaton can be obtained.

*Other alphabets:* The notion of functional stream derivative is defined for functions on streams over arbitrary alphabets. Having an algebra of stream functions of which the operators are defined by means of stream differential equations, seems to be the basis for the application of our construction. We expect that further applications can be found in other algebras, for instance as used in the context of functional programming languages.

*Digital circuits:* It is well-known how to construct a digital circuit out of a binary Mealy automaton (see for instance [14] for a classical reference). Thus one can start with an algebraic specification of a binary stream function, then use (a suitable implementation of) our construction to obtain a Mealy automaton, and finally use standard techniques to construct a corresponding digital circuit out of the Mealy automaton. All in all, this would give a fully automated path from the algebraic specification of (certain) arithmetical functions to a corresponding hardware implementation.

Note that minimality results for binary Mealy automata are also relevant for circuits. In particular, the minimal number of memory cells (one bit registers) that a circuit needs in order to implement the functional specification, is given by the 2-log of the number of states of the corresponding minimal automaton. In this way, one can use, for instance, Theorem 5.1 to compute lowerbounds for the numbers of registers needed to implement arithmetic functions (not much is known about this, cf. [12,22]).

*Implementation:* For algebras such as that of the 2-adic numbers, the construction of Mealy automata from algebraic expressions can in many cases be automated. For this, one has to be more precise about the distinction between syntax and semantics than we have been here. In particular, one has to reduce algebraic expressions to normal form, in order to decide whether functional derivatives result in new states or not. At present, our construction is being further analysed and implemented for certain classes of functions from binary arithmetic, by Helle Hvid Hansen and David de Oliveira Costa.

*Related work:* In [7], derivatives of rational expressions are used to obtain (Moore style) automata over an arbitrary (input) alphabet. The same paper also shows how to apply the construction to obtain Mealy automata over arbitrary input alphabet and the *fixed* output alphabet  $2 = \{0, 1\}$ . There are two major differences: (i) our approach applies to *arbitrary* output alphabets; (ii) we use (algebraic expressions denoting) stream functions rather than rational expressions (denoting formal languages) to specify the behaviour of our automata. The latter point is also relevant for the case that (both inputs and) outputs are binary, a case which *is* covered by [7]. As we have seen, many operators from binary arithmetic can be easily specified in the 2-adic algebra of bitstreams, whereas the use of rational expressions (over the alphabet 2) would be at best inconvenient and often impossible.

There does not seem to exist much literature on the construction of Mealy automata from algebraically specified stream functions. Most approaches use logical specification formalisms (see for instance [6,11]). There one starts with a logically specified *relation* (sometimes called stream requirement) on input and output streams, and the goal is to find at least one Mealy automaton (out of many possible ones) whose behaviour satisfies the requirement. Here we start with an algebraically specified *function* and construct the unique (minimal) automaton that implements it.

## Acknowledgement

Many thanks to Helle Hvid Hansen for various comments on earlier versions of this paper.

## References

- [1] V. Antimirov. *Partial derivatives of regular expressions and finite automata constructions*. *Theoretical Computer Science*, 155:291–319, 1996.

- [2] F. Arbab. *Reo: a channel-based coordination model for component composition*. *Mathematical Structures in Computer Science*, 14:329–366, 2004.
- [3] F. Arbab, C. Baier, F. de Boer, J. Rutten, and M. Sirjani. *Synthesis of Reo circuits*. In *Proceedings of Coordination 2005*, volume 3454 of *LNCS*, pages 236–251. Springer, 2005.
- [4] F. Arbab, C. Baier, J. Rutten, and M. Sirjani. *Modeling component connectors in Reo by constraint automata*. In *Proceedings of FOCLASA 2003*, volume 97 of *ENTCS*. Elsevier, 2003. Extended version to appear in *Science of Computer Prog.*, 2005.
- [5] F. Arbab and J.J.M.M. Rutten. *A coinductive calculus of component connectors*. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *Proceedings of WADT 2002*, volume 2755 of *LNCS*, pages 35–56. Springer, 2003.
- [6] A. Aziz, F. Balarin, R. Brayton, and A. Sangiovanni. *Sequential synthesis using SIS*. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 19(10):1149–1162, 2000.
- [7] J.A. Brzozowski. *Derivatives of regular expressions*. *Journal of the ACM*, 11(4):481–494, 1964.
- [8] Christian Choffrut. *Minimizing subsequential transducers: a survey*. *Theoretical Computer Science*, 292:131–143, 2003.
- [9] S. Eilenberg. “Automata, languages and machines (Vol. A)”. Pure and applied mathematics. Academic Press, 1974.
- [10] M. Goresky and A. Klapper. *Feedback shift registers, combiners with memory, and 2-adic span*. *Journal of Cryptology*, 10:111–147, 1997.
- [11] T. Henzinger, S. Krishnan, O. Kupferman, and F. Mang. *Synthesis of uninitialized systems*. In *Proceedings of ICALP’02*, volume 2380 of *Lecture Notes in Computer Science*. Springer, 2002.
- [12] E.L. Johnson and M.A. Karim. “Digital design”. PWS Publishers, 1987.
- [13] N. Koblitz.  *$p$ -adic Numbers,  $p$ -adic Analysis, and Zeta-Functions*, volume 58 of “Graduate Texts in Mathematics”. Springer-Verlag, 1977.
- [14] Z. Kohavi. “Switching and finite automata theory”. McGraw-Hill, 1978.
- [15] S. Lombardy and J. Sakarovitch. *Derivatives of rational expressions with multiplicity*. In *Proceedings of MFCS’02*, volume 2420 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
- [16] S. Lombardy and J. Sakarovitch. *Derivatives of rational expressions with multiplicity*. *Theoretical Computer Science*, 332:141–177, 2005.
- [17] G.N. Raney. *Sequential functions*. *J. Assoc. Comput. Mach.*, 3:177–180, 1958.
- [18] Christophe Reutenauer. *Subsequential functions: characterizations, minimizations, examples*. In *Internat. Meeting of Young Computer Scientists*, volume 464 of *LNCS*, pages 62–79. Springer, 1990.
- [19] J.J.M.M. Rutten. *Automata, power series, and coinduction: taking input derivatives seriously (extended abstract)*. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *Proceedings of ICALP’99*, volume 1644 of *LNCS*, pages 645–654, 1999.
- [20] J.J.M.M. Rutten. *Behavioural differential equations: a coinductive calculus of streams, automata, and power series*. *Theoretical Computer Science*, 308(1):1–53, 2003. Fundamental Study.
- [21] J.J.M.M. Rutten. *Algebra, bitstreams, and circuits*. In *Proceedings of the Dresden Conference 2004 (AAA68)*, volume 16 of *Contributions to General Algebra*, pages 231–250. Verlag Johannes Heyn, 2005.
- [22] J. Vuillemin. *On circuits and numbers*. *IEEE Transactions on Computers*, 43(8):868–879, 1994.