Stream Differential Equations:
Specification Formats and Solution Methods

Helle Hvid Hansen Clemens Kupke Jan Rutten

August 8, 2014

Abstract

Streams, or infinite sequences, are infinite objects of a very simple type, yet they have a rich theory partly due to their ubiquity in mathematics and computer science. Stream differential equations are a coinductive method for specifying streams and stream operations, and their theory has been developed in many papers over the past two decades. In this paper we present a survey of the many results in this area. Our focus is on the classification of different formats of stream differential equations, their solution methods, and the classes of streams they can define. Moreover, we describe in detail the connection between the so-called syntactic solution method and abstract GSOS.

1 Introduction

Streams, or infinite sequences, are infinite objects of a very simple type, yet they have a rich theory partly due to their ubiquity. Streams occur as numerical expansions, data sequences, formal power series, limit sequences, dynamic system behaviour, formal languages, ongoing computations, and much more.

Defining the *stream derivative* of a stream $\sigma = (\sigma(0), \sigma(1), \sigma(2), \ldots)$ by

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

and the *initial value* of σ by $\sigma(0)$, one can develop a *calculus of streams* in close analogy with classical calculus in mathematical analysis. Notably we can, using the notions of stream derivative and initial value, specify streams by means of *stream differential equations*.

For instance, the stream differential equation $\sigma(0) = 1,\ \sigma' = \sigma$ has the stream $\sigma = (1, 1, 1, \ldots)$ as its unique solution, and the equation $\sigma(0) = 1,\ \sigma' = \sigma + \sigma$, where $+$ is the elementwise addition of two streams, defines the stream $(2^0, 2^1, 2^2, \ldots)$. Similarly, we can specify stream functions. For example, $f(\sigma)(0) = \sigma(0),\ f(\sigma)' = f(\sigma'')$ (this time using second-order derivatives) defines the function $f(\sigma) = (\sigma(0), \sigma(2), \sigma(4), \ldots)$. In these examples, it is easy to see that the stream differential equations have a unique solution.

But how about $\tau(0) = 0,\ \tau' = f(\tau)$? A moment's thought reveals that this equation has several solutions, e.g. $\tau = (0, 0, 0, \ldots)$ and $\tau = (0, 0, 1, 1, 1, \ldots)$. But what is the difference between this equation and the previous ones? How can we ensure the existence of unique solutions? Which classes of streams can be defined using a finite amount of information? These questions have been studied by several authors in many different contexts in recent years, and have led to notions such as rational streams, context-free streams, and new insights into automatic and regular sequences.

In this paper, we present an overview of the current state-of-the-art in formats and solution methods for stream differential equations. The theoretical basis for stream dfferential equations is given by coalgebra [51], but our aim is to give an elementary and self-contained overview. We consider our contribution to be a unified and uniform presentation of results which are collected from many different sources. As modest new insights, we mention the results on the expressiveness of non-standard formats in Section 8. Another contribution which can be considered new, is the detailed analysis of the connection between the syntactic method and abstract GSOS (in Section 9) This connection is rather obvious to readers familiar with abtract GSOS, but probably less so to the uninitiated reader.

Overview: We start by giving an informal introduction to stream differential equations in Section 2, and in Section 3 we provide some basic definitions regarding automata and stream calculus. In Section 4, we describe a general solution method for a large class of well-formed stream differential equations.

Next, in Sections 5 through 8, we shall study in more detail various types of stream differential equations, each corresponding to a specification format. We describe computational solution methods, and characterise the automata and the classes of streams that these families of stream differential equations can define. The following little table contains some representative examples, corresponding to Sections 5 through 8:

initial value:	derivative:	solution:	type of equation:
$\sigma(0) = 1$	$\sigma' = \sigma$	$(1, 1, 1, \ldots)$	simple
$\sigma(0) = 1$	$\sigma' = \sigma + \sigma$	$(2^0, 2^1, 2^2, \ldots)$	linear
$\sigma(0) = 1$	$\sigma' = \sigma \times \sigma$	Catalan numbers	context-free/algebraic
$\sigma(0) = 1$	$\frac{d}{dX}(\sigma) = \sigma$	$\left(\frac{1}{0!}, \frac{1}{1!}, \frac{1}{2!}, \frac{1}{3!}, \frac{1}{4!}, \ldots\right)$	non-standard

(For the definition of the convolution product $\times$ see (6) in Section 2; the non-standard derivative $\frac{d}{dX}$ is defined in Example 8.1.)

In Section 9, a more general, categorical perspective on the theory of stream differential equations is presented. In particular, this section places streams (and also languages) and automata in a more general context of algebras, coalgebras and so-called distributive laws. Note that this is the only section that requires some basic knowledge of category theory. In Section 10, we briefly discss connections with other methods for representing streams, such as recurrence relations, generating functions and so on.

Related work: Here we mention the most important origins of the results in this paper. A more extensive discussion of related work is found in Section 11. Stream differential equations [55] came about as a special instance of behavioural differential equations, for formal power series, which were introduced in [53]. Motivation came from the coalgebraic perspective on infinite data structures, in which streams are a canonical example, but also from work on language dervatives in classical automata theory, notably [14] and [15]. The idea of developing a calculus of streams in close analogy to analysis was further inspired by the work on classical calculus in coinductive form in [49]. The classification of stream differential equations into the families of simple, linear and context-free systems stems from our joint work with Marcello Bonsangue and Joost Winter, in [12] and [66], on classifications of behavioural differential equations for streams, languages and formal power series. The results on non-standard stream calculus come from [37, 36], and the examples on automatic and regular sequences from [38] and [25], respectively.

Acknowledgements: It should be clear from the many references to the literature that our paper builds on the work of many others. We are, in particular, much indebted to

Marcello Bonsangue and Joost Winter for many years of excellent collaboration on stream differential equations. A large part of the work presented here was developed in joint work with them. We are also grateful to many other colleagues, with whom we have worked together in different ways on ideas relating to streams, including: Henning Basold, Filippo Bonchi, Joerg Endrullis, Herman Geuvers, Clemens Grabmayer, Dimitri Hendriks, Bart Jacobs, Bartek Klin, Jan-Willem Klop, Dorel Lucanu, Larry Moss, Milad Niqui, Grigore Rosu, Jurriaan Rot, Alexandra Silva, Joost Winter, Hans Zantema.

Contents

2 Stream Differential Equations

In this section, we present several examples of *stream differential equations (SDEs)* and their solutions. For now the purpose is to get familiarised with the notation of SDEs. Detailed proofs and solution methods are presented later.

We start by introducing notation and basic definitions on streams.

2.1 Basic definitions

A stream over a given set A is a function $\sigma\colon \mathbb{N} \to A$ from the natural numbers to A, which we will sometimes write as

$$\sigma = (\sigma(0), \sigma(1), \sigma(2), \ldots)$$

The set of all streams over A is denoted by

$$A^\omega = \{\sigma \mid \sigma\colon \mathbb{N} \to A\}$$

Given a stream $\sigma \in A^\omega$, we define the *initial value* of σ as $\sigma(0)$, and the *derivative* of σ as the stream $\sigma' = (\sigma(1), \sigma(2), \sigma(3), \ldots)$. For $a \in A$ and $\sigma \in A^\omega$, we define $a : \sigma = (a, \sigma(0), \sigma(1), \sigma(2), \ldots)$. Higher order derivatives $\sigma^{(n)}$ are defined inductively for all $n \in \mathbb{N}$ by:

$$\sigma^{(0)} = \sigma \qquad \sigma^{(n+1)} = (\sigma^{(n)})'$$

Initial value and derivative are also known as *head* and *tail*, respectively.

2.2 Simple examples

Stream differential equations define a stream in terms of its initial value and its derivative(s). As a first elementary example, consider

$$\sigma(0) = 1, \quad \sigma' = \sigma. \tag{1}$$

which has the stream $\mathsf{ones} = (1, 1, 1, \ldots)$ as the unique solution.

For a slightly more interesting example consisting of two SDEs over two stream variables, consider

$$\begin{array}{ll} \sigma(0) = 1, & \sigma' = \tau \\ \tau(0) = 0, & \tau' = \sigma \end{array} \tag{2}$$

whose solution is $\sigma = (1, 0, 1, 0, \ldots)$ and $\tau = (0, 1, 0, 1, \ldots)$.

In the above SDEs, derivatives are specified by another stream variable. Such SDEs are called simple, and in Section 5, we will characterise the class of streams that can be specified by finite systems of simple equations. More generally, we will consider SDEs involving not only variables, but also operations on streams.

2.3 Stream operations

We illustrate how to define stream operations, and at the same time introduce a bit of stream calculus. Stream calculus is usually defined for streams over the real numbers $\mathbb{R}$, but most definitions hold for more general data domains A.

Consider the set A^ω of streams over a ring $(A, +, -, \cdot, 0, 1)$. The stream differential equation:

$$(\sigma + \tau)(0) \;=\; \sigma(0) + \tau(0), \qquad (\sigma + \tau)' \;=\; \sigma' + \tau' \tag{3}$$

defines the element-wise addition of two streams, that is, for all $\sigma, \tau \in A^\omega$,

$$(\sigma + \tau)(n) = \sigma(n) + \tau(n) \qquad \text{for all } n \in \mathbb{N}. \tag{4}$$

(Note that we use the same symbol to denote addition in A and addition of streams. The typing should be clear from the context.)

Similarly, one can define the element-wise multiplication with a scalar $a \in A$ with the SDE:

$$(a \cdot \sigma)(0) = a \cdot \sigma(0), \qquad (a \cdot \sigma)' = a \cdot \sigma'. \tag{5}$$

where $\cdot$ denotes multiplication in A. It follows that, for all $\sigma \in A^\omega$,

$$(a \cdot \sigma)(n) = a \cdot \sigma(n) \qquad \text{for all } n \in \mathbb{N}.$$

Clearly, any element-wise operation on streams can be defined in a similar manner. An example of a non-element-wise operation is the *convolution product* of streams given explicitly by:

$$(\sigma \times \tau)(n) \;=\; \sum_{k=0}^{n} \sigma(k) \cdot \tau(n-k) \qquad \text{for all } n \in \mathbb{N} \tag{6}$$

which is defined by the SDE

$$(\sigma \times \tau)(0) = \sigma(0) \cdot \tau(0), \qquad (\sigma \times \tau)' = (\sigma' \times \tau) + ([\sigma(0)] \times \tau') \tag{7}$$

where for $a \in A$,

$$[a](0) = a, \qquad [a]' = [0] \tag{8}$$

One can show that the convolution product is commutative if and only if the multiplication in A is commutative.

Using the convolution product, and taking $A = \mathbb{Z}$ we can form the following SDE:

$$\sigma(0) = 1, \qquad \sigma' = \sigma \times \sigma \tag{9}$$

It defines the stream $\sigma = (1, 1, 2, 5, 14, 42, 132, 429, 1430, \ldots)$ of Catalan numbers, cf. [12].

When A is a field such as the reals $\mathbb{R}$, some streams σ have an inverse σ^{-1} with respect to convolution product, that is, $\sigma \times \sigma^{-1} = \sigma^{-1} \times \sigma = [1]$. By taking initial value on both sides we find that the inverse should satisfy $(\sigma \times \sigma^{-1})(0) = 1$ and hence by the definition of convolution product, $\sigma^{-1}(0) = 1/\sigma(0)$ which exists only if $\sigma(0) \neq 0$ in A. Similarly, taking derivatives on both sides and rearranging leads to the following SDE:

$$\sigma^{-1}(0) = 1/\sigma(0), \qquad (\sigma^{-1})' = [-1/\sigma(0)] \times \sigma' \times \sigma^{-1} \tag{10}$$

2.4 Higher-order examples

Just as with classical differential equations, SDEs can also be higher-order. For instance, the second-order SDE

$$\sigma(0) = 0,\ \sigma'(0) = 1, \qquad \sigma'' = \sigma' + \sigma \tag{11}$$

(with $+$ as defined above) defines the stream of Fibonacci numbers $\sigma = (0, 1, 1, 2, 3, 5, 8, \ldots)$. An n-order SDE can always be represented as a system of n first-order SDEs. For example, the Fibonacci stream is equivalently defined as the solution for σ in

$$\begin{array}{ll} \sigma(0) = 0, & \sigma' = \tau \\ \tau(0) = 1, & \tau' = \tau + \sigma \end{array} \tag{12}$$

A similar example is given by

$$\begin{array}{rclrcl} \sigma(0) & = & 1, & \sigma' & = & \sigma \\ \tau(0) & = & 0, & \tau' & = & \tau + \sigma \end{array} \tag{13}$$

We know already that ones is a solution for σ (cf. equation (1)). Hence (13) is equivalent to

$$\tau(0) = 0, \qquad \tau' = \tau + \mathsf{ones} \tag{14}$$

which has $\tau = (0, 1, 2, 3, 4, \ldots)$ as its unique solution.

A slightly more involved example is given by the stream γ of Hamming numbers (or regular numbers) which consists of natural numbers of the form $2^i 3^j 5^k$ for $i, j, k \geq 0$ in increasing order (cf. [16, 67]). The first part of γ looks like $(1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, \ldots)$. The stream γ can be defined by the following SDE:

$$\gamma(0) = 1, \qquad \gamma' = (2 \cdot \gamma) \parallel ((3 \cdot \gamma) \parallel (5 \cdot \gamma)) \tag{15}$$

where $n \cdot \gamma$, $n \in \mathbb{N}$, is the scalar multiplication defined as in (5) and $\parallel$ is the *merge* operator defined by

$$(\sigma \| \tau)(0) = \begin{cases} \sigma(0) & \text{if } \sigma(0) < \tau(0) \\ \tau(0) & \text{if } \sigma(0) \geq \tau(0) \end{cases} \qquad (\sigma \| \tau)' = \begin{cases} \sigma' \| \tau & \text{if } \sigma(0) < \tau(0) \\ \sigma' \| \tau' & \text{if } \sigma(0) = \tau(0) \\ \sigma \| \tau' & \text{if } \sigma(0) > \tau(0) \end{cases} \tag{16}$$

That is, $\parallel$ merges two streams into one by taking smallest initial values first, and removing duplicates.

Many more examples of stream differential equations inspired by analysis, arithmetic and combinatorics can be found in [52, 53, 55].

3 Stream Automata and Stream Calculus

We will show how one can prove the existence of unique solutions to SDEs by using the notions of stream automata and coinduction.

3.1 Stream Automata and coinduction

Streams can be represented by so-called stream automata. A *stream automaton* (with output in A) is a pair $\langle X, s\rangle$ where X is a set (called the state space, or the carrier) and $s = \langle o, d\rangle \colon X \to A \times X$ is a function that maps each $x \in X$ to a pair consisting of an output value $o(x) \in A$ and a unique next state $d(x) \in X$. We will write $x \xrightarrow{a} y$ when $o(x) = a$ and $d(x) = y$. A small example of a stream automaton is given in Figure 1. (In categorical

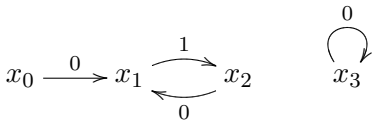


Figure 1: Stream automaton with $X = \{x_0, x_1, x_2, x_3\}$ and $A = \{0, 1\}$.

terms, a stream automaton is a *coalgebra* for the functor F on Set defined by $FX = A \times X$, cf. Section 9.)

Intuitively, a state x in a stream automaton $\langle X, \langle o, d\rangle\rangle$ represents the stream of outputs that can be observed by following the transitions starting in x:

$$(o(x), o(d(x)), o(d(d(x)), o(d(d(d(x)))), \ldots)$$

This stream is called the *(observable) behaviour of* x. For example, the behaviour of the state x_0 in Figure 1 is the stream $(0, 1, 0, 1, \ldots)$. We will now characterise behaviour using the notion of homomorphism and finality.

A *homomorphism of stream automata* is a function between state spaces that preserves outputs and transitions. Formally, a function $f \colon X_1 \to X_2$ is a homomorphism from $\langle X_1, \langle o_1, d_1\rangle\rangle$ to $\langle X_2, \langle o_2, d_2\rangle\rangle$ iff the following diagram commutes:

$$\begin{array}{ccc} X_1 & \xrightarrow{f} & X_2 \\ {\scriptstyle \langle o_1, d_1\rangle}\downarrow & & \downarrow{\scriptstyle \langle o_2, d_2\rangle} \\ A \times X_1 & \xrightarrow{\mathrm{id}_A \times f} & A \times X_2 \end{array}$$

where id_A denotes the identity map on A. Equivalently: for all $x \in X_1$,

$$o_1(x) = o_2(f(x)) \text{ and } f(d_1(x)) = d_2(f(x)).$$

The set of streams A^ω is itself a stream automaton under the map $\zeta \colon \sigma \mapsto \langle \sigma(0), \sigma'\rangle$, and it is moreover *final* which means that for any stream automaton $\langle o, d\rangle \colon X \to A \times X$ there is a unique stream homomorphism $[\![-]\!] \colon X \to A^\omega$ (called the *final map*) into $\langle A^\omega, \zeta\rangle$:

$$\begin{array}{ccc} X & \xrightarrow{\exists ![\![-]\!]} & A^\omega \\ {\scriptstyle \forall \langle o, d\rangle}\downarrow & & \downarrow{\scriptstyle \zeta} \\ A \times X & \xrightarrow{\mathrm{id}_A \times [\![-]\!]} & A \times A^\omega \end{array}$$

By the commutativity of the above diagram, we find that

$$[\![x]\!] = (o(x), o(d(x)), o(d(d(x)), o(d(d(d(x)))), \ldots)$$

is indeed the observable behaviour of $x \in X$. The final map $[\![-]\!]$ is therefore often referred to as the *behaviour map*. Note that by uniqueness, the final map from $\langle A^\omega, \zeta \rangle$ to itself must be the identity homomorphism, that is, for all $\sigma \in A^\omega$:

$$[\![\sigma]\!] = \sigma. \tag{17}$$

It can now easily be verified that for the stream automaton in Figure 1, the behaviour map is:

$$\begin{array}{rcl} [\![x_0]\!] = [\![x_2]\!] & = & (0,1,0,1,\ldots) \\ [\![x_1]\!] & = & (1,0,1,0,\ldots) \\ [\![x_3]\!] & = & (0,0,0,0,\ldots) \end{array}$$

The universal property of the final stream automaton yields a coinductive definition principle and a coinductive proof principle, both are often referred to as *coinduction*. In this paper we will make extensive use of both.

A map $f\colon X \to A^\omega$ is said to be *defined by coinduction*, if it is obtained as the unique homomorphism into the final stream automaton. In practice, such an f is obtained by equipping X with a suitable stream automaton structure and using the finality of $\langle A^\omega, \zeta \rangle$.

A *proof by coinduction* is based on the notion of bisimulation. Let $\langle X_1, \langle o_1, d_1 \rangle \rangle$ and $\langle X_2, \langle o_2, d_2 \rangle \rangle$ be stream automata. A relation $R \subseteq X_1 \times X_2$ is a *stream bisimulation* if for all $\langle x_1, x_2 \rangle \in R$:

$$o_1(x_1) = o_2(x_2) \quad \text{and} \quad d_1(x_1)\, R\, d_2(x_2). \tag{18}$$

Two states x_1 and x_2 are said to be *bisimilar*, written $x_1 \sim x_2$, if they are linked by some stream bisimulation. We list a few well known facts about stream bisimulations, cf. [53].

Lemma 3.1 *Let $\langle X_1, s_1 \rangle$ to $\langle X_2, s_2 \rangle$ be stream automata.*

1. *If $R_i \subseteq X_1 \times X_2$, $i \in I$, are stream bisimulations then $\bigcup_{i \in I} R_i$ is a bisimulation.*
2. *The bisimilarity relation $\sim \subseteq X_1 \times X_2$ is the largest bisimulation between $\langle X_1, s_1 \rangle$ to $\langle X_2, s_2 \rangle$.*
3. *If $f\colon X_1 \to X_2$ is a stream homomorphism, then its graph $R = \{\langle x, f(x) \rangle \mid x \in X_1\}$ is a stream bisimulation.*

The main result regarding bisimilarity is stated in the following theorem.

Theorem 3.2 *For all stream automata $\langle X_i, s_i \rangle$ and all $x_i \in X_i$, $i = 1, 2$,*

$$x_1 \sim x_2 \quad \iff \quad [\![x_1]\!] = [\![x_2]\!]$$

Consequently, by (17)*, for all streams $\sigma, \tau \in A^\omega$,*

$$\sigma \sim \tau \quad \iff \quad \sigma = \tau$$

From Theorem 3.2 we get the coinductive proof principle: to prove that two streams are equal it suffices to show that they are bisimilar.

Finally, we also need the notions of subautomaton and minimal automaton. A stream automaton $\langle Y, t\rangle$ is a *subautomaton* of $\langle X, s\rangle$ if $Y \subseteq X$ and the inclusion map $\iota\colon Y \to X$ is a stream homomorphism, which means that $t = s|_Y$. Given a stream automaton $\langle X, s\rangle$, the *subautomaton generated by* $Y_0 \subseteq X$ is the subautomaton $\langle Y, t\rangle$ obtained by closing Y_0 under transitions. A stream automaton $\langle X, s\rangle$ is *minimal* if the behaviour map $[\![-]\!]\colon X \to A^\omega$ is injective. Note that due to Theorem 3.2, every subautomaton of the final stream automaton is minimal.

Referring to the automaton in Figure 1, an example of a stream bisimulation is given by $\{(x_0, x_2), (x_1, x_1), (x_2, x_2)\}$. It follows from Theorem 3.2 that $[\![x_0]\!] = [\![x_2]\!]$.

3.2 Stream Calculus

In this short section, we introduce some further preliminaries on stream calculus that we will be using in the remainder of the paper. As we have seen already, if A has algebraic structure, then we can use it to define algebraic structure on A^ω. For example, if A is a commutative ring, then also

$$(A^\omega, +, -, \times, [0], [1])$$

is a commutative ring, cf. [53, Thm.4.1]. Similarly, if A is a field, then A^ω is a vector space over A with the operations of scalar multiplication and addition.

Table 1 summarises the SDEs defining the stream calculus operations on A^ω most of which were already introduced in Section 2.3. The fact that these SDEs have unique solutions will follow from the results in Section 4.

derivative:	initial value:	name:
$[a]' = [0]$	$[a](0) = a$	constant $[a], a \in A$
$(\sigma + \tau)' = \sigma' + \tau'$	$(\sigma + \tau)(0) = \sigma(0) + \tau(0)$	sum
$(a \cdot \sigma)' = a \cdot (\sigma')$	$(a \cdot \sigma)(0) = a \cdot \sigma(0)$	scalar multiplication
$(-\sigma)' = -(\sigma')$	$(-\sigma)(0) = -\sigma(0)$	minus
$(\sigma \times \tau)' = (\sigma' \times \tau) + ([\sigma(0)] \times \tau')$	$(\sigma \times \tau)(0) = \sigma(0) \cdot \tau(0)$	convolution product
$(\sigma^{-1})' = -[\sigma(0)^{-1}] \times \sigma' \times \sigma^{-1}$	$(\sigma^{-1})(0) = \sigma(0)^{-1}$	convolution inverse

Table 1: Operations of stream calculus. Recall (from section 2.3) that convolution inverse is defined only if $\sigma(0)$ is an invertible element of A. In the case A is a field this is equivalent with $\sigma(0) \neq 0$, and as usual, we will often write $\frac{\sigma}{\tau}$ for $\sigma \times \tau^{-1}$.

We further add to our stream calculus the constant stream

$$\mathsf{X} = (0, 1, 0, 0, 0, \ldots) \quad \text{defined by} \quad \mathsf{X}(0) = 0, \qquad \mathsf{X}' = [1].$$

Multiplication by X acts as "stream integration" (seen as an inverse to stream derivative) since

$$(\mathsf{X} \times \sigma)' = \sigma$$

This follows from the fact that, for all $\sigma \in A^\omega$,

$$\mathsf{X} \times \sigma \;=\; \sigma \times \mathsf{X} \;=\; (0, \sigma(0), \sigma(1), \sigma(2), \ldots)$$

This leads to the very useful *fundamental theorem of stream calculus* [53, Thm. 5.1].

Theorem 3.3 *For every* $\sigma \in A^\omega$, $\sigma = [\sigma(0)] + (\mathsf{X} \times \sigma')$.

Proof. For all σ, we have:

$$\begin{aligned} \sigma &= (\sigma(0), \sigma(1), \sigma(2), \ldots) \\ &= (\sigma(0), 0, 0, 0, \ldots) + (0, \sigma(1), \sigma(2), \sigma(3), \ldots) \\ &= [\sigma(0)] + (\mathsf{X} \times \sigma') \end{aligned}$$

QED

We conclude this section by an enhancement of the bisimulation proof method.

Definition 3.4 (bisimulation-up-to) *Let* Σ *denote a collection of stream operations. A relation* $R \subseteq A^\omega \times A^\omega$ *is a* (stream) bisimulation-up-to-Σ *if for all* $(\sigma, \tau) \in R$*:*

$$\sigma(0) = \tau(0) \qquad \textit{and} \qquad (\sigma', \tau') \in \bar{R},$$

where $\bar{R} \subseteq A^\omega \times A^\omega$ *is the smallest relation such that*

1. $R \subseteq \bar{R}$
2. $\{ \langle \sigma, \sigma \rangle \mid \sigma \in A^\omega \} \subseteq \bar{R}$
3. $\bar{R}$ *is closed under the (element-wise application of) operations in* Σ*. (For instance, if* Σ *contains addition and* $\langle \alpha, \beta \rangle, \langle \gamma, \delta \rangle \in \bar{R}$ *then* $\langle \alpha + \gamma,\ \beta + \delta \rangle \in \bar{R}$*.)*

We write $\sigma \sim_\Sigma \tau$ *if there exists a bisimulation-up-to-*Σ *containing* $\langle \sigma, \tau \rangle$*.*

Theorem 3.5 (coinduction-up-to) *Let* Σ *be a subset of the stream calculus operations from Table 1. We have:*

$$\sigma \sim_\Sigma \tau \quad \Rightarrow \quad \sigma = \tau \tag{19}$$

Proof. If R is a bisimulation-up-to-Σ, then $\bar{R}$ can be shown to be a bisimulation relation by structural induction on its definition. The theorem then follows by Theorem 3.2. QED

4 Syntactic Method

The examples of the previous section illustrate the general approach to defining streams and stream operations by systems of SDEs. In this section, we discuss a general method for showing that many such systems of SDEs have a unique solution. Because the method associates with each such system of SDEs a set of terms, we call it *syntactic*. As we shall see, the method will work for all systems of SDEs that satisfy a rather general condition on their (syntactic) shape. Later, in Sections 5, 6 and 7, we shall treat various specific families of SDEs in more detail, amongst others, by characterising the classes of streams that they define. An earlier version of the material in this section is found in [36].

The basic idea of the syntactic method is as follows. Given a signature Σ of operations and a set X (of generators), let $T_\Sigma(X)$ denote the set of all Σ-terms over X. A system of SDEs, one for each operation in Σ, yields an inductive definition of a stream automaton

$\langle o, d\rangle \colon T_\Sigma(A^\omega) \to A \times T_\Sigma(A^\omega)$ which has terms as states. The stream solutions are obtained via coinduction:

$$\begin{array}{ccc} T_\Sigma(A^\omega) & \overset{[\![-]\!]}{\dashrightarrow} & A^\omega \\ {\scriptstyle \langle o,d\rangle}\downarrow & & \downarrow{\scriptstyle \zeta} \\ A \times T_\Sigma(A^\omega) & \dashrightarrow & A \times A^\omega \end{array}$$

The behaviour map $[\![-]\!]$ thus yields for each term $t \in T_\Sigma(A^\omega)$ a stream $[\![t]\!] \in A^\omega$, in other words, it defines an algebra (of signature Σ) on the set of streams.

Example 4.1 *In order to define the sequence of natural numbers as in* (13)*, we take* $A = \mathbb{N}$ *and* $\Sigma = \{\mathsf{ones}, \mathsf{nats}, +\}$ *where* ones *and* nats *are 0-ary operations (constants), and* $+$ *is binary. The associated (infinite) system of SDEs consists of all defining SDEs put together:*

$$\begin{array}{ll} \mathsf{ones}(0) = 1, & \mathsf{ones}' = \mathsf{ones} \\ \mathsf{nats}(0) = 0, & \mathsf{nats}' = \mathsf{nats} + \mathsf{ones} \\ (s+t)(0) = s(0) + t(0), & (s+t)' = s' + t' \qquad \forall s, t \in T_\Sigma(A^\omega) \end{array}$$

The shape of the SDEs seen so far are all instances of the general format called *stream GSOS*, cf. [35]. Informally stated, a system of SDEs is in the stream GSOS format if for all k-ary operations $\underline{f}$ in Σ, the SDE defining $\underline{f}$ has the shape:

$$\underline{f}(\sigma_1, \ldots, \sigma_n)(0) = a, \quad \underline{f}(\sigma_1, \ldots, \sigma_n)' = t$$

where $a \in A$ depends only on $\sigma_1(0), \ldots, \sigma_k(0)$, and t is a Σ-term over $\sigma_1, \ldots, \sigma_n, \sigma_1', \ldots, \sigma_n'$ that depends only on $\sigma_1(0), \ldots, \sigma_k(0)$.

To see how things can go wrong when straying from the GSOS format, consider the following SDE (for the signature Σ containing a single constant c):

$$\mathsf{c}(0) = 1, \quad \mathsf{c}' = \mathsf{c}' \tag{20}$$

This SDE has no unique solution, since any stream starting with a 1 is a solution, and indeed (20) is not in the GSOS format. The reason is that the derivative of c should be defined as a term $t \in T_\Sigma(\emptyset)$, and $\mathsf{c}' \notin T_\Sigma(\emptyset) = \{\mathsf{c}\}$ (since the derivative operation is not part of the signature). Moreover, note that it is not possible to extend the signature with the derivative operation. This follows from the fact that all stream operations defined in the GSOS format are *causal* (as we will see in Proposition 4.10), a property which the derivative operation lacks. We return to causal operations in section 4.3.

In the remainder of this section we present and prove the correctness of the syntactic method for SDEs in the stream GSOS format. This result follows from more general insights in the theory of bialgebras and abstract GSOS, cf. [7, 35, 60], and we give a brief summary of this more abstract, categorical presentation in Section 9. In the current section, we wish to present a self-contained, elementary proof of this fact. Most of the results in this section are also found in [36].

4.1 Terms and algebras

A signature Σ is a collection of operation symbols $\underline{f}$, each of which has an arity k. Nullary operations (with arity 0) are called constants, and unary operations are called functions. We

write Σ_k for the set of k-ary operations in Σ. The set of Σ-terms over a set X (of generators) is denoted by $T_\Sigma(X)$, and defined inductively as the least set T that contains X and is closed under the following formation rule: if $t_1, \ldots, t_k$ are in T and $\underline{f}$ is in Σ_k, $k \in \mathbb{N}$, then $\underline{f}(t_1, \ldots, t_k)$ is in T.

A Σ-*algebra* $\langle X, \alpha\rangle$ consists of a carrier set X and a collection of maps $\alpha = \{f_\alpha \colon X^k \to X \mid \underline{f} \in \Sigma_k, k \in \mathbb{N}\}$ containing for each k-ary operation $\underline{f} \in \Sigma$, a map $f_\alpha \colon X^k \to X$ interpreting $\underline{f}$. A *homomorphism of* Σ*-algebras* from $\langle X, \alpha\rangle$ to $\langle Y, \beta\rangle$ is a function $h \colon X \to Y$ which respects the algebra structure, i.e., for all $\underline{f} \in \Sigma_k$, $k \in \mathbb{N}$, and all $x_1, \ldots, x_k \in X$: $h(f_\alpha(x_1, \ldots, x_k)) = f_\beta(h(x_1), \ldots, h(x_k))$.

For any X, the set $T_\Sigma(X)$ of Σ-terms over X is a Σ-algebra $\langle T_\Sigma(X), \gamma_\Sigma\rangle$ where γ_Σ is given by construction of terms. In fact, it is the so-called *free* Σ*-algebra over* X which means that if $\langle Y, \alpha\rangle$ is a Σ-algebra and $h \colon X \to Y$ is a function mapping generators to elements in Y, then there is a unique homomorphism $h^* \colon \langle T_\Sigma(X), \gamma_\Sigma\rangle \to \langle Y, \alpha\rangle$ extending h which is defined inductively by:

$$\begin{array}{rcll} h^*(x) & = & h(x) & \text{for all } x \in X, \\ h^*(\underline{f}(t_1, \ldots, t_k)) & = & f_\alpha(h^*(t_1), \ldots, h^*(t_k)) & \text{for all } \underline{f} \in \Sigma_k, k \in \mathbb{N}. \end{array}$$

Note that every homomorphism $g \colon \langle T_\Sigma(X), \gamma_\Sigma\rangle \to \langle Y, \alpha\rangle$ is determined by its action on the generators X. In other words, there is a 1-1 correspondence between homomorphisms $\langle T_\Sigma(X), \gamma_\Sigma\rangle \to \langle Y, \alpha\rangle$ and maps $X \to Y$. In particular, a Σ-algebra $\langle X, \alpha\rangle$ corresponds uniquely to a homomorphism $\overline{\alpha} \colon \langle T_\Sigma(X), \gamma_\Sigma\rangle \to \langle X, \alpha\rangle$ (by taking $\overline{\alpha}$ to be the homomorphic extension id_X^*). We call the homomorphism $\overline{\alpha} \colon \langle T_\Sigma(X), \gamma_\Sigma\rangle \to \langle X, \alpha\rangle$ the *interpretation of* Σ*-terms induced by* α.

Terms come equipped with the standard notion of substitution. A *substitution* is a homomorphism $s \colon T_\Sigma(X) \to T_\Sigma(Y)$. For a term $t \in T_\Sigma(X)$ over variables $x_1, \ldots, x_k \in X$ and a substitution s for which $s(x_i) = s_i$ for $i = 1, \ldots, k$, we write $t[s_i/x_i]_{i \le k}$ for the result of applying substitution s to t.

4.2 Stream GSOS definitions

Let Σ be an arbitrary, but fixed signature. A *stream GSOS definition for* $\underline{f} \in \Sigma_k$, $k \in \mathbb{N}$, is formally a pair $\langle o_{\underline{f}}, d_{\underline{f}}\rangle$ (defining "initial value" $o_{\underline{f}}$ and "derivative" $d_{\underline{f}}$ of $\underline{f}$) where

$$\begin{aligned} o_{\underline{f}} &: A^k \to A \\ d_{\underline{f}} &: A^k \to T_\Sigma(\{x_1, \ldots, x_k, y_1, \ldots, y_k\}) \end{aligned}$$

The pair $\langle o_{\underline{f}}, d_{\underline{f}}\rangle$ corresponds to a stream differential equation:

$$\begin{array}{rcl} \underline{f}(\sigma_1, \ldots, \sigma_k)(0) & = & o_{\underline{f}}(\sigma(0), \ldots, \sigma_k(0)) \\ \underline{f}(\sigma_1, \ldots, \sigma_k)' & = & d_{\underline{f}}(\sigma(0), \ldots, \sigma_k(0))[\sigma_i/x_i, \sigma_i'/y_i]_{i \le k} \end{array} \tag{21}$$

If $d_{\underline{f}}(a_1, \ldots, a_k)$ does not contain any of the x_i variables, then we say that $\langle o_{\underline{f}}, d_{\underline{f}}\rangle$ is in the *simple SOS format* for $\underline{f}$.

A *stream GSOS definition for* Σ is a set $\mathcal{D}$ of stream GSOS definitions $\langle o_{\underline{f}}, d_{\underline{f}}\rangle$, one for each $\underline{f} \in \Sigma$. If for all $\underline{f}$ in Σ, $\langle o_{\underline{f}}, d_{\underline{f}}\rangle$ is in the simple SOS format, then we say that $\mathcal{D}$ is a simple stream SOS definition for Σ.

Example 4.2 (GSOS definition of arithmetic operations) *Let $A = \mathbb{R}$. The SDEs defining addition and convolution product on streams of real numbers in* (3) *and* (7) *are equivalent to the following GSOS definition. Take as signature $\Sigma = \{\underline{+}, \underline{\times}\} \cup \{\underline{[a]} \mid a \in \mathbb{R}\}$, where $\underline{+}$ and $\underline{\times}$ are binary operation symbols and $\underline{[a]}$ is a constant symbol, for all $a \in \mathbb{R}$, and let $\langle o_{\underline{[a]}}, d_{\underline{[a]}}\rangle$, $\langle o_{\underline{+}}, d_{\underline{+}}\rangle$, $\langle o_{\underline{\times}}, d_{\underline{\times}}\rangle$ be defined as follows:*

$$\begin{array}{lllll} o_{\underline{[a]}} = a & & d_{\underline{[a]}} = [0] & & \text{for all } a \in \mathbb{R}, \\ o_{\underline{+}}(a,b) = a+b, & & d_{\underline{+}}(a,b) = y_1 \mathbin{\underline{+}} y_2, & & \\ o_{\underline{\times}}(a,b) = a \cdot b, & & d_{\underline{\times}}(a,b) = \begin{cases} y_1 \mathbin{\underline{\times}} x_2 & \text{if } a = 0 \\ (y_1 \mathbin{\underline{\times}} x_2) \mathbin{\underline{+}} (\underline{[a]} \mathbin{\underline{\times}} y_2) & \text{if } a \neq 0 \end{cases} & & \end{array}$$

where $+$ and $\cdot$ on the right-hand sides of o-definitions denote addition and multiplication of real numbers.

A *solution* of a stream GSOS definition $\mathcal{D}$ for Σ is a Σ-algebra $\langle A^\omega, \alpha\rangle$ on the set of streams which respects $\mathcal{D}$, that is, for all $\underline{f} \in \Sigma$, $k \in \mathbb{N}$,

$$\begin{array}{lll} f_\alpha(\sigma_1, \ldots, \sigma_k)(0) & = & o_{\underline{f}}(\sigma_1(0), \ldots, \sigma_k(0)) \\ f_\alpha(\sigma_1, \ldots, \sigma_k)' & = & \overline{\alpha}(d_{\underline{f}}(\sigma_1(0), \ldots, \sigma_k(0))[\sigma_i/x_i, \sigma_i'/y_i]_{i \leq k}) \end{array} \tag{22}$$

This definition, in fact, says that α is a solution if the induced interpretation $\overline{\alpha}$ is a homomorphism not only of algebras, but also of stream automata. We will make this precise below.

We will now prove that every stream GSOS definition $\mathcal{D}$ has a unique solution. Using the correspondence between Σ-algebras on A^ω and interpretations $T_\Sigma(A^\omega) \to A^\omega$, we obtain a candidate solution by coinduction by observing that a stream GSOS definition $\mathcal{D}$ yields a stream automaton structure on $T_\Sigma(A^\omega)$.

Definition 4.3 (Syntactic stream automaton) *Let $\mathcal{D}$ be a stream GSOS definition for a signature Σ. The* syntactic stream automaton for $\mathcal{D}$ *is the map $\langle o_\mathcal{D}, d_\mathcal{D}\rangle \colon T_\Sigma(A^\omega) \to A \times T_\Sigma(A^\omega)$ defined inductively as follows: For all $\sigma \in A^\omega$,*

$$o_\mathcal{D}(\sigma) = \sigma(0), \qquad d_\mathcal{D}(\sigma) = \sigma'$$

and for all $k \in \mathbb{N}$, $\underline{f} \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma(A^\omega)$,

$$\begin{array}{lll} o_\mathcal{D}(\underline{f}(t_1, \ldots, t_k)) & = & o_{\underline{f}}(o_\mathcal{D}(t_1), \ldots, o_\mathcal{D}(t_k)) \\ d_\mathcal{D}(\underline{f}(t_1, \ldots, t_k)) & = & d_{\underline{f}}(o_\mathcal{D}(t_1), \ldots, o_\mathcal{D}(t_k))[t_i/x_i, d_\mathcal{D}(t_i)/y_i]_{i \leq k} \end{array}$$

Example 4.4 (Syntactic arithmetic automaton) *In the syntactic automaton of the GSOS definition from Example 4.2 (taking $A = \mathbb{R}$), we have, for example, the following states and transitions. Let $\sigma = (2, 0, 0, \ldots)$, $\tau = (1, 1, 1, \ldots)$, $\delta = (1, 0, 0, \ldots)$ and $\rho = (0, 0, 0, \ldots)$.*

$$\begin{array}{rcl} \sigma \times (\tau + \delta) & \xrightarrow{4} & (\sigma' \times (\tau + \delta)) + ([2] \times (\tau' + \delta')) \\ & & = \\ & & (\rho \times (\tau + \delta)) + ([2] \times (\tau + \rho)) \\ [5] \times \sigma & \xrightarrow{10} & ([0] \times \sigma) + ([5] \times [0]) \end{array}$$

The final homomorphism of stream automata from $\langle T_\Sigma(A^\omega), \langle o_\mathcal{D}, d_\mathcal{D}\rangle\rangle$ is denoted by $[\![-]\!]_\mathcal{D}$, i.e.,

$$\begin{array}{ccc} T_\Sigma(A^\omega) & \xrightarrow{\quad [\![-]\!]_\mathcal{D} \quad} & A^\omega \\ {\scriptstyle \langle o_\mathcal{D}, d_\mathcal{D}\rangle}\downarrow & & \downarrow{\scriptstyle \zeta} \\ A \times T_\Sigma(A^\omega) & \xrightarrow{\mathrm{id}_A \times [\![-]\!]_\mathcal{D}} & A \times A^\omega \end{array} \tag{23}$$

and we let $\alpha_\mathcal{D}$ be the Σ-algebra on A^ω obtained by restricting $[\![-]\!]_\mathcal{D}$ to terms of depth 1. That is, $\alpha_\mathcal{D} = \{f_\mathcal{D} \colon (A^\omega)^k \to A^\omega \mid \underline{f} \in \Sigma_k, k \in \mathbb{N}\}$ where

$$f_\mathcal{D}(\sigma_1, \ldots, \sigma_k) = [\![\underline{f}(\sigma_1, \ldots, \sigma_k)]\!]_\mathcal{D} \tag{24}$$

The definition of the syntactic stream automaton ensures that the following fundamental result holds.

Lemma 4.5 (Bisimilarity is a congruence) *On the syntactic stream automaton $\langle T_\Sigma(A^\omega), \langle o_\mathcal{D}, d_\mathcal{D}\rangle\rangle$, bisimilarity is a congruence, that is, for all terms $g \in T_\Sigma(Z)$ over some set of variables $Z = \{z_1, \ldots, z_n\}$, and all terms $s_1, \ldots, s_n, u_1, \ldots, u_n \in T_\Sigma(A^\omega)$,*

$$\forall j = 1, \ldots, n : s_j \sim u_j \quad \Rightarrow \quad g[s_j/z_j]_{j\le n} \sim g[u_j/z_j]_{j\le n}$$

Proof. We define relations $\{R_m\}_{m\in\mathbb{N}}$ on $T_\Sigma(A^\omega)$ inductively by $R_0 := \sim$ (the bisimilarity relation on $T_\Sigma(A^\omega)$) and for $m \ge 1$, R_{m+1} is defined by the following congruence rule:

$$\frac{s_1 \, R_m \, u_1 \quad \cdots \quad s_n \, R_m \, u_n}{g[s_j/z_j]_{j\le n} \, R_{m+1} \, g[u_j/z_j]_{j\le n}} (g \in T_\Sigma(Z)) \tag{25}$$

where $Z = \{z_1, \ldots, z_n\}$. Note that $R_m \subseteq R_{m'}$ for all $m \le m'$. Let $R = \bigcup_{m\in\mathbb{N}} R_m$. We show that R is a bisimulation. More precisely, we show by induction on m that

$$\forall t, v \in T_\Sigma(A^\omega) : t \, R_m \, v \;\Rightarrow\; [o_\mathcal{D}(t) = o_\mathcal{D}(v) \text{ and } d_\mathcal{D}(t) \, R \, d_\mathcal{D}(v)] \tag{26}$$

For convenience, we use the shorthand notation $t[s] := t[s_j/z_j]_{j\le n}$ and $t[u] := t[u_j/z_j]_{j\le n}$ for any term $t \in T_\Sigma(Z)$.

The base case ($m = 0$) is immediate since $R_0 = \sim$ and $\sim \subseteq R$. For the induction step $(m+1)$, suppose $s_1 \, R_m \, u_1$, … , $s_n \, R_m \, u_n$ and $g \in T_\Sigma(Z)$. We show by subinduction on the term structure of g that

$$o_\mathcal{D}(g[s]) = o_\mathcal{D}(g[u]) \quad \text{and} \quad d_\mathcal{D}(g[s]) \, R \, d_\mathcal{D}(g[u]) \tag{27}$$

For $g = z_j \in Z$, it follows that $\langle g[s], g[u]\rangle = \langle s_j, u_j\rangle \in R_m$ and hence (27) holds by the main induction hypothesis (for m).

For $g = \underline{f}(t_1, \ldots, t_k)$, by subinduction hypothesis, we have for all $i = 1, \ldots, k$:

$$o_\mathcal{D}(t_i[s]) = o_\mathcal{D}(t_i[u]) \quad \text{and} \quad d_\mathcal{D}(t_i[s]) \; R \; d_\mathcal{D}(t_i[u]).$$

We now check the subinduction claim (27) for g.
Outputs are equal:

$$\begin{array}{lll} & o_\mathcal{D}(\underline{f}(t_1, \ldots, t_k)[s]) & \\ = & o_{\underline{f}}(o_\mathcal{D}(t_1[s]), \ldots, o_\mathcal{D}(t_k[s])) & \text{(def. } o_\mathcal{D}\text{)} \\ = & o_{\underline{f}}(o_\mathcal{D}(t_1[u]), \ldots, o_\mathcal{D}(t_k[u])) & \text{(sub-I.H.)} \\ = & o_\mathcal{D}(\underline{f}(t_1, \ldots, t_k)[u]) & \text{(def. } o_\mathcal{D}\text{)} \end{array}$$

Next states are related: First, for notational convenience, let w denote the term that specifies the next state for $\underline{f}$, i.e.,

$$\begin{array}{rcl} w & := & d_{\underline{f}}(o_{\mathcal{D}}(t_1[s]), \ldots, o_{\mathcal{D}}(t_k[s])) \\ & \overset{\text{sub-I.H.}}{=} & d_{\underline{f}}(o_{\mathcal{D}}(t_1[u]), \ldots, o_{\mathcal{D}}(t_k[u])) \end{array}$$

From the definition of R it follows that

$$t_i[s]\, R_{m+1}\, t_i[u] \qquad \text{for all } i = 1, \ldots, k$$

and from the sub-induction hypothesis, it follows that

$$d_{\mathcal{D}}(t_i[s])\, R\, d_{\mathcal{D}}(t_i[u]) \qquad \text{for all } i = 1, \ldots, k.$$

hence there is some $M \in \mathbb{N}$ such that

$$t_i[s]\, R_M\, t_i[u], \quad d_{\mathcal{D}}(t_i[s])\, R_M\, d_{\mathcal{D}}(t_i[u]) \qquad \text{for all } i = 1, \ldots, k.$$

By the definition of R, we then have

$$w[t_i[s]/x_i, d_{\mathcal{D}}(t_i[s])/y_i]_{i \leq k}\ R_{M+1}\ w[t_i[u]/x_i, d_{\mathcal{D}}(t_i[u])/y_i]_{i \leq k} \tag{28}$$

and hence

$$\begin{array}{rll} & d_{\mathcal{D}}(\underline{f}(t_1, \ldots, t_k)[s]) & \\ = & w[t_i[s]/x_i, d_{\mathcal{D}}(t_i[s])/y_i]_{i \leq k} & \text{(def. } d_{\mathcal{D}}\text{)} \\ R & w[t_i[u]]/x_i, d_{\mathcal{D}}(t_i[u]])/y_i]_{i \leq k} & \text{(by (28))} \\ = & d_{\mathcal{D}}(\underline{f}(t_1, \ldots, t_k)[u]) & \text{(def. } d_{\mathcal{D}}\text{).} \end{array}$$

This concludes the subinduction on g, and hence also the main induction for m. QED

The map $[\![-]\!]_{\mathcal{D}}$ is by definition a stream homomorphism. We now show that it is also an algebra homomorphism.

Lemma 4.6 ($[\![-]\!]_{\mathcal{D}}$ is algebra homomorphism) *Let $\mathcal{D}$ be a stream GSOS definition for a signature Σ, and $\alpha_{\mathcal{D}}$ be the Σ-algebra on A^ω defined in* (24) *of Definition 4.3. The term interpretation $\overline{\alpha_{\mathcal{D}}}$ induced by $\alpha_{\mathcal{D}}$ is precisely $[\![-]\!]_{\mathcal{D}}$. Consequently, $[\![-]\!]_{\mathcal{D}}$ is a morphism of Σ-algebras.*

Proof. Let $\alpha_{\mathcal{D}} = \{f_{\mathcal{D}} \colon (A^\omega)^k \to A^\omega \mid \underline{f} \in \Sigma_k, k \in \mathbb{N}\}$ be defined as in (24). We show by induction on the term structure that for all $t \in T_\Sigma(A^\omega)$:

$$\overline{\alpha_{\mathcal{D}}}(t) = [\![t]\!]_{\mathcal{D}} \tag{29}$$

For $t = \sigma \in A^\omega$, we clearly have that $\overline{\alpha_{\mathcal{D}}}(\sigma) = \sigma = [\![\sigma]\!]_{\mathcal{D}}$. For $k \in \mathbb{N}$, $\underline{f} \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma(A^\omega)$, we have

$$\begin{array}{rcll} \overline{\alpha_{\mathcal{D}}}(\underline{f}(t_1, \ldots, t_k)) & = & f_{\mathcal{D}}(\overline{\alpha_{\mathcal{D}}}(t_1), \ldots, \overline{\alpha_{\mathcal{D}}}(t_k)) & \text{(def. } \overline{\alpha_{\mathcal{D}}}\text{)} \\ & = & f_{\mathcal{D}}([\![t_1]\!]_{\mathcal{D}}, \ldots, [\![t_k]\!]_{\mathcal{D}}) & \text{(I.H.)} \\ & = & [\![\underline{f}([\![t_1]\!]_{\mathcal{D}}, \ldots, [\![t_k]\!]_{\mathcal{D}})]\!]_{\mathcal{D}} & \text{(def. } f_{\mathcal{D}}\text{)} \\ & = & [\![\underline{f}(t_1, \ldots, t_k)]\!]_{\mathcal{D}} & \end{array}$$

where the last equality holds because $[\![-]\!]_{\mathcal{D}}$ identifies bisimilar states, and bisimilarity of $\underline{f}([\![t_1]\!]_{\mathcal{D}}, \ldots, [\![t_k]\!]_{\mathcal{D}})$ and $\underline{f}(t_1, \ldots, t_k)$ follows from Lemma 4.5 (bisimilarity is a congruence), and the fact that for all $t \in T_\Sigma(A^\omega)$,

$$t \sim [\![t]\!]_{\mathcal{D}} \tag{30}$$

since $[\![-]\!]_{\mathcal{D}}$ is a stream homomorphism. QED

We now characterise the solutions to $\mathcal{D}$ as being those maps α whose induced interpretation is a stream homomorphism.

Proposition 4.7 *Let $\mathcal{D}$ be a stream GSOS definition for a signature Σ. For all Σ-algebras $\langle A^\omega, \alpha\rangle$, α is a solution of $\mathcal{D}$ if and only if $\overline{\alpha}\colon T_\Sigma(A^\omega) \to A^\omega$ is a stream automaton homomorphism from $\langle T_\Sigma(A^\omega), \langle o_{\mathcal{D}}, d_{\mathcal{D}}\rangle\rangle$ to the final stream automaton $\langle A^\omega, \zeta\rangle$.*

Proof. Let α be a solution of $\mathcal{D}$. We show that $\overline{\alpha}$ is a homomorphism of stream automata by induction on the term structure. The base case is immediate, since by definition $\overline{\alpha}(\sigma)(0) = \sigma(0) = o_{\mathcal{D}}(\sigma)$ and $\overline{\alpha}(\sigma)' = \sigma' = d_{\mathcal{D}}(\sigma)$. For the inductive step, let $k \in \mathbb{N}$, $\underline{f} \in \Sigma_k$. We have

$$\begin{array}{rcll}
\overline{\alpha}(\underline{f}(t_1, \ldots, t_k))(0) & = & f_\alpha(\overline{\alpha}(t_1), \ldots, \overline{\alpha}(t_k))(0) & \text{(def. } \overline{\alpha}\text{)} \\
& = & o_{\underline{f}}(\overline{\alpha}(t_1)(0), \ldots, \overline{\alpha}(t_k)(0)) & (\alpha \text{ is solution}) \\
& = & o_{\underline{f}}(o_{\mathcal{D}}(t_1), \ldots, o_{\mathcal{D}}(t_k)) & \text{(by I.H.)} \\
& = & o_{\mathcal{D}}(\underline{f}(t_1, \ldots, t_k)) & \text{(def. } o_{\mathcal{D}}\text{)}
\end{array}$$

and

$$\begin{array}{rcll}
 & \overline{\alpha}(\underline{f}(t_1, \ldots, t_k))' & & \\
= & f_\alpha(\overline{\alpha}(t_1), \ldots, \overline{\alpha}(t_k))' & & \text{(def. } \overline{\alpha}\text{)} \\
= & \overline{\alpha}(d_{\underline{f}}(\overline{\alpha}(t_1)(0), \ldots, \overline{\alpha}(t_k)(0))[\overline{\alpha}(t_i)/x_i, \overline{\alpha}(t_i)'/y_i]_{i\leq k}) & & (\alpha \text{ is solution}) \\
= & \overline{\alpha}(d_{\underline{f}}(o_{\mathcal{D}}(t_1), \ldots, o_{\mathcal{D}}(t_k))[\overline{\alpha}(t_i)/x_i, \overline{\alpha}(d_{\mathcal{D}}(t_i))/y_i]_{i\leq k}) & & \text{(I.H.)} \\
= & \overline{\alpha}(d_{\underline{f}}(o_{\mathcal{D}}(t_1), \ldots, o_{\mathcal{D}}(t_k))[t_i/x_i, d_{\mathcal{D}}(t_i)/y_i]_{i\leq k}) & & (*) \\
= & \overline{\alpha}(d_{\mathcal{D}}(\underline{f}(t_1, \ldots, t_k))) & & \text{(def. } d_{\mathcal{D}}\text{)}
\end{array}$$

where $(*)$ holds since nested applications of $\overline{\alpha}$ are "flattened" into one outermost application which interprets the entire term.

For the converse, assume that $\overline{\alpha}$ is a homomorphism of stream automata. Then in particular, for all $k \in \mathbb{N}$, $\underline{f} \in \Sigma_k$, and all $\sigma_1, \ldots, \sigma_k \in A^\omega$,

$$\overline{\alpha}(\underline{f}(\sigma_1, \ldots, \sigma_k))(0) = o_{\mathcal{D}}(\underline{f}(\sigma_1, \ldots, \sigma_k)) \tag{31}$$

$$\overline{\alpha}(\underline{f}(\sigma_1, \ldots, \sigma_k))' = \overline{\alpha}(d_{\mathcal{D}}(\underline{f}(\sigma_1, \ldots, \sigma_k))) \tag{32}$$

It follows that

$$\begin{array}{rcll}
f_\alpha(\sigma_1, \ldots, \sigma_k)(0) & = & \overline{\alpha}(\underline{f}(\sigma_1, \ldots, \sigma_k))(0) & \text{(def. } \overline{\alpha}\text{)} \\
& = & o_{\mathcal{D}}(\underline{f}(\sigma_1, \ldots, \sigma_k)) & \text{(by (31))} \\
& = & o_{\underline{f}}(\sigma_1(0), \ldots, \sigma_k(0)) & \text{(def. } o_{\mathcal{D}}\text{)}
\end{array}$$

and

$$\begin{array}{rcll}
 & f_\alpha(\sigma_1, \ldots, \sigma_k)' & & \\
= & \overline{\alpha}(\underline{f}(\sigma_1, \ldots, \sigma_k))' & & \text{(def. } \overline{\alpha}\text{)} \\
= & \overline{\alpha}(d_{\mathcal{D}}(\underline{f}(\sigma_1, \ldots, \sigma_k))) & & \text{(by (32))} \\
= & \overline{\alpha}(d_{\underline{f}}(o_{\mathcal{D}}(\sigma_1), \ldots, o_{\mathcal{D}}(\sigma_k))[\sigma_i/x_i, d_{\mathcal{D}}(\sigma_i)/y_i]_{i\leq k}) & & \text{(def. } d_{\mathcal{D}}\text{)} \\
= & \overline{\alpha}(d_{\underline{f}}(\sigma_1(0), \ldots, \sigma_k(0))[\sigma_i/x_i, \sigma_i'/y_i]_{i\leq k}) & & \text{(def. } o_{\mathcal{D}} \text{ and } d_{\mathcal{D}}\text{)}
\end{array}$$

which proves that α is indeed a solution of $\mathcal{D}$. QED
Finally, we can put everything together.

Theorem 4.8 *Let $\mathcal{D}$ be a stream GSOS definition for a signature Σ. The unique solution of $\mathcal{D}$ is the Σ-algebra $\langle A^\omega, \alpha_\mathcal{D}\rangle$ that corresponds to the term interpretation given by the final stream homomorphism $[\![-]\!]_\mathcal{D} \colon T_\Sigma(A^\omega) \to A^\omega$ of the syntactic stream automaton.*

Proof. By Lemma 4.6, $\overline{\alpha_\mathcal{D}} = [\![-]\!]_\mathcal{D}$, hence by Proposition 4.7, $\alpha_\mathcal{D}$ is a solution to $\mathcal{D}$. The uniqueness of $\alpha_\mathcal{D}$ follows from the uniqueness of $[\![-]\!]_\mathcal{D}$ and the 1-1 correspondence between Σ-algebras $\langle X, \alpha\rangle$ and term interpretations $\overline{\alpha} \colon T_\Sigma(A^\omega) \to A^\omega$. QED

Example 4.9 (Solutions arithmetic automaton) *Consider the final map $[\![-]\!] = [\![-]\!]_\mathcal{D}$ for the GSOS definition $\mathcal{D}$ from Example 4.2 (taking again $A = \mathbb{R}$, and $\sigma = (2,0,0,\ldots)$, $\tau = (1,1,1,\ldots)$, $\delta = (1,0,0,\ldots)$, $\rho = (0,0,0,\ldots)$). We find that*

$$\begin{aligned} [\![\sigma \times (\tau + \delta)]\!] &= [\![\sigma]\!] \times ([\![\tau]\!] + [\![\delta]\!]) \\ &= (4,2,2,2,\ldots) \\ [\![(\rho \times (\tau + \delta)) + ([2] \times (\tau + \rho))]\!] &= ([\![\rho]\!] \times ([\![\tau]\!] + [\![\delta]\!])) + ([\![[2]]\!] \times ([\![\tau]\!] + [\![\rho]\!])) \\ &= (2,2,2,\ldots) \end{aligned}$$

which confirms that $[\![-]\!]$ respects the transition from $\sigma \times (\tau + \delta)$. Similarly, we find that the following transition in the syntactic automaton

$$[5] \times \sigma \quad \xrightarrow{10} \quad ([0] \times \sigma) + ([5] \times [0])$$

is mapped by $[\![-]\!]$ to the following transition in $\langle \mathbb{R}^\omega, \zeta\rangle$

$$(10,0,0,0,\ldots) \xrightarrow{10} (0,0,0,\ldots).$$

4.3 Causal stream operations

In conclusion of this section, we show that stream GSOS definitions exactly define the so-called *causal* stream operations, that is, operations for which the n-th value of the result stream depends only on the first n values of the argument stream(s). For a formal definition, we use the following notation. For $\sigma, \tau \in A^\omega$ and $n \in \mathbb{N}$, we write $\sigma \equiv_n \tau$ if for all $j \leq n$, $\sigma(j) = \tau(j)$. A k-ary stream operation $f \colon (A^\omega)^k \to A^\omega$ is *causal* if for all $\sigma_i, \tau_i \in A^\omega, i = 1,\ldots,k$,

$$\forall i \leq k : \sigma_i \equiv_n \tau_i \qquad \Rightarrow \qquad f(\sigma_1,\ldots,\sigma_k) \equiv_n f(\tau_1,\ldots,\tau_k)$$

Let Γ_k denote the set of all causal k-ary stream operations $f \colon (A^\omega)^k \to A^\omega$. The elements of Γ_k are exactly the behaviours of *(k-ary) Mealy machines* which are maps of type $m \colon X \to (A \times X)^{A^k}$. Mealy machines and causal stream functions are treated in detail in [26, 57]. We give a brief recap here. For all $f \in \Gamma_k$ and all $a_1,\ldots,a_k \in A$, we define the notion of *Mealy output* $f[a_1,\ldots,a_k]$ and *Mealy derivative* $f_{(a_1,\ldots,a_k)}$ of f as follows. For all $\sigma_1,\ldots,\sigma_k \in A^\omega$,

$$\begin{aligned} f[a_1,\ldots,a_k] &= f(a_1{:}\sigma_1,\ldots,a_k{:}\sigma_k)(0) \\ f_{(a_1,\ldots,a_k)}(\sigma_1,\ldots,\sigma_k) &= f(a_1{:}\sigma_1,\ldots,a_k{:}\sigma_k)' \end{aligned} \tag{33}$$

Note that since f is causal, it follows that $f_{(a_1,\ldots,a_k)} \in \Gamma_k$ and that $f[a_1,\ldots,a_k]$ is well-defined, as it does not depend on $\sigma_1,\ldots,\sigma_k$. We define a Mealy machine structure $\gamma\colon \Gamma_k \to (A\times\Gamma^k)^{A^k}$ by

$$\gamma(f)(a_1,\ldots,a_k) = \langle f[a_1,\ldots,a_k], f_{(a_1,\ldots,a_k)}\rangle, \tag{34}$$

In fact, $\langle \Gamma_k, \gamma\rangle$ is a final Mealy machine, cf. [26, 57].

Proposition 4.10 *If $f\colon (A^\omega)^k \to A^\omega$ is stream GSOS definable, then f is causal.*

Proof. Suppose that f is stream GSOS definable, that is, f is one of the operations in the solution $\alpha_{\mathcal{D}}$ for some stream GSOS definition $\mathcal{D}$. The proof follows essentially from the fact that for all $n \in \mathbb{N}$, $\equiv_n$ is a congruence, that is, for all $t \in T_\Sigma(Z)$, $Z = \{z_1,\ldots,z_l\}$, and all $\sigma_i, \tau_i \in A^\omega$, $i = 1,\ldots,l$:

$$\text{for all } i \le l : \sigma_i \equiv_n \tau_i \qquad \Rightarrow \qquad [\![t[\sigma_i/z_i]_{i\le l}]\!]_{\mathcal{D}} \equiv_n [\![t[\tau_i/z_i]_{i\le l}]\!]_{\mathcal{D}} \tag{35}$$

which can be shown by double induction on n and the structure of t. We refer to [36] for details. QED

Conversely, any causal stream operation can be defined by a (potentially very large) stream definition.

Proposition 4.11 *If $f\colon (A^\omega)^k \to A^\omega$ is causal, then f is stream GSOS definable.*

Proof. We define a stream definition $\mathcal{D}_{\mathsf{C}}$ for the signature $\Sigma_{\mathsf{C}} = \{\underline{f} \mid f\colon (A^\omega)^k \to A^\omega \text{ causal}, k \in \mathbb{N}\}$, by including, for each k-ary function symbol $\underline{f} \in \Sigma_{\mathsf{C}}$, the equation

$$\begin{array}{lcll} o_{\underline{f}}(a_1,\ldots,a_k) & = & f[a_1,\ldots,a_k] & \in A \\ d_{\underline{f}}(a_1,\ldots,a_k) & = & \underline{f_{(a_1,\ldots,a_k)}}(y_1,\ldots,y_k) & \in T_\Sigma(\{y_1,\ldots,y_k\}) \end{array} \tag{36}$$

We show that $\Gamma = \bigcup_{k\in\mathbb{N}} \Gamma_k$ is a solution to $\mathcal{D}_{\mathsf{C}}$ where Γ is the Σ_{C}-algebra in which each symbol $\underline{f}$ is interpreted as f. For $f \in \Gamma_k$, we have

$$\begin{array}{lcl} f(\sigma_1,\ldots,\sigma_k)(0) & = & f[\sigma_1(0),\ldots,\sigma_k(0)] \\ & = & o_{\underline{f}}(\sigma_1(0),\ldots,\sigma_k(0)) \\ f(\sigma_1,\ldots,\sigma_k)' & = & f_{(\sigma_1(0),\ldots,\sigma_k(0))}(\sigma_1',\ldots,\sigma_k') \\ & = & \overline{\alpha}(\underline{f_{(\sigma_1(0),\ldots,\sigma_k(0))}}(y_1,\ldots,y_k)[\sigma_i'/y_i]_{i\le k}) \end{array}$$

which shows that Γ is a solution to $\mathcal{D}_{\mathsf{C}}$. QED

Remark 4.12 *Note that in* (36) *the derivative term $d_{\underline{f}}(a_1,\ldots,a_k)$ only uses the y_i-variables, i.e. the derivatives of the arguments, and not the arguments themselves (i.e. the x_i-variables). This means that all causal stream operations are definable by a (possibly infinite) simple SOS specification.*

Theorem 4.13 *Let $f\colon (A^\omega)^k \to A^\omega$ be a stream operation. We have: f is causal if and only if f is stream GSOS definable.*

4.4 Causality and productivity

Every stream GSOS defined operation is *productive*, meaning that by successively computing output and derivative using the SDEs we can construct the entire stream in the limit.

A well known example of a stream operation that is not causal is the operation

$$\mathsf{even}(\sigma) \quad = \quad (\sigma(0), \sigma(2), \sigma(4), \ldots)$$

which can be defined by the following SDE:

$$\mathsf{even}(\sigma)(0) \quad = \quad \sigma(0), \qquad \mathsf{even}(\sigma)' \quad = \quad \mathsf{even}(\sigma'')$$

If σ is given by a productive definition, then also $\mathsf{even}(\sigma)$ is productive. However, it is easy to give a SDE using even which is not productive:

$$\sigma(0) = 0, \qquad \sigma' = \mathsf{even}(\sigma) \tag{37}$$

One sees the problem when we try to compute initial value and derivatives. The first two steps are fine:

1.	$\sigma(0) = 0,$	$\sigma' = \mathsf{even}(\sigma)$
2.	$\mathsf{even}(\sigma)(0) = \sigma(0) = 0,$	$\mathsf{even}(\sigma)' = \mathsf{even}(\sigma'')$

But when we try to compute the initial value of $\mathsf{even}(\sigma'')$, we get:

$$\mathsf{even}(\sigma'')(0) = \sigma''(0) = ?$$

which does not yield a value. The SDE (37) has several solutions, e.g. $\sigma = [0] = (0, 0, 0, \ldots)$ or $\sigma = 0 : 0 : \mathsf{ones} = (0, 0, 1, 1, 1, \ldots)$, but no unique one.

Productivity of stream definitions in a term rewriting context have been studied closely in [20, 18].

5 Simple Specifications

In sections 5-7, we will characterise the classes of streams, i.e. subsets of A^ω, that arise as the solutions to finite systems of SDEs over varying algebraic structures/signatures.

We start by defining the most simple type of systems of SDEs. Let A be an arbitrary set. A *simple equation system* over a set (of variables) $X = \{x_i \mid i \in I\}$ is a collection of SDEs, one for each $x_i \in X$, of the form

$$x_i(0) \quad = \quad a_i, \qquad x_i' \quad = \quad y_i;$$

where $a_i \in A$ and $y_i \in X$ for all $i \in I$. A simple equation system corresponds to a map $e\colon X \to A \times X$, i.e., to a stream automaton with state space X. A *solution of* $e\colon X \to A \times X$ is an assignment $h\colon X \to A^\omega$ of variables to streams that preserves the equations: $h(x_i) = a_i$ and $h(x_i)' = h(y_i)$ for all $i \in I$. This holds exactly when the following diagram commutes:

$$\begin{array}{ccc} X & \xrightarrow{\quad h \quad} & A^\omega \\ {\scriptstyle e}\downarrow & & \downarrow{\scriptstyle \zeta} \\ A \times X & \xrightarrow{\mathrm{id}_A \times h} & A \times A^\omega \end{array}$$

In other words, solutions are stream homorphisms from (X, e) to the final stream automaton. By coinduction, solutions to simple equation systems exist and are unique. We will also say that a stream $\sigma \in A^\omega$ is a solution of e if $\sigma = h(x)$ for some $x \in X$, in which case we call e a *specification of* σ.

Note that a simple equation system (stream automaton) $\langle X, e\rangle$ can be seen as a stream definition over the signature Σ which contains a constant for each $x \in X$, and no further operation symbols. Hence $T_\Sigma(A^\omega) = X$, and it follows that the syntactic solution from Theorem 4.8 coincides with the direct solution by coinduction.

We call a simple equation system $e\colon X \to A \times X$ finite if X is finite. The SDEs in equations (1) and (2) are examples of finite simple equation systems. Note that any stream σ is the solution of the *infinite* simple equation system $e\colon \mathbb{N} \to A \times \mathbb{N}$ defined by: $n(0) = \sigma(n)$ and $n' = n + 1$, for all $n \in \mathbb{N}$.

The solutions of finite simple equation systems are exactly the behaviours of finite stream automata, which are precisely the *eventually periodic streams*. This is easy to prove. We state the result explicitly to make clear the analogue with the results on linear and context-free specifications that will be discussed in Sections 6 and 7.

Proposition 5.1 *The following are equivalent for all streams $\sigma \in A^\omega$.*

1. *σ is the solution of a finite simple equation system.*
2. *σ generates a finite subautomaton of the final stream automaton.*
3. *σ is eventually periodic, i.e., $\sigma^{(k)} = \sigma^{(n)}$ for some $k < n \in \mathbb{N}$.*

Proof. $1 \Rightarrow 2$: Let $h\colon X \to A^\omega$ be a solution of the finite $e\colon X \to A \times X$ and $\sigma = h(x)$ for some $x \in X$. The subautomaton generated by σ is contained in the image $h(X)$ which is finite, since X is finite.

$2 \Rightarrow 3$: The subautomaton generated by σ has as its state set $\{\sigma^{(k)} \mid k \in \mathbb{N}\}$ which is finite by assumption. Consequently, there are $k, n \in \mathbb{N}$ such that $\sigma^{(k)} = \sigma^{(n)}$ and $k < n$.

$3 \Rightarrow 1$: Assume that $\sigma^{(k)} = \sigma^{(n)}$ for $k < n \in \mathbb{N}$. Let $X = \{x_0, \ldots, x_{n-1}\}$ and define $e\colon X \to A \times X$ by

$$x_i = \sigma(i), \quad x_i' = x_{i+1} \text{ if } i < n-1, \text{ else } x_k \text{ for all } i = 0, \ldots, n-1.$$

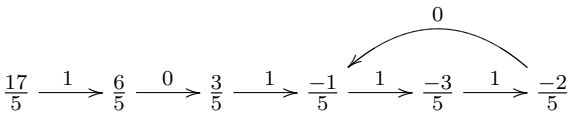
Now $\sigma = h(x_0)$ where h is the unique solution of e. QED

Eventually periodic streams constitute some of the simplest infinite objects that have a finite representation. Such finite representations make it possible to compute with and reason about infinite objects. We list a few examples.

Example 5.2 (Rational numbers in binary) *Let $2 = \{0, 1\}$ denote the set of bits. Rational numbers with odd denominator, that is, elements of $\mathbb{Q}_{odd} = \{q = \frac{n}{2m+1} \mid n, m \in \mathbb{Z}\}$, can be represented as eventually periodic bitstreams. The representation $B\colon \mathbb{Q}_{odd} \to 2^\omega$ is obtained by coinduction via the following stream automaton structure on $\mathbb{Q}_{odd}$:*

$$o(q) = n \bmod 2, \qquad d(q) = (q - o(q))/2$$

For example, the finitary representation of the number $\frac{17}{5}$ can be found by computing output and derivatives leading to the following stream automaton:

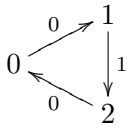


Hence, $B(\frac{17}{5}) = 101(110)^\omega$. Such base 2 expansions allow for efficient implementations of arithmetic operations, cf. [28].

Example 5.3 (Regular languages over one-letter alphabet) *A bitstream $\sigma \in 2^\omega$ corresponds to a language $L \subseteq \mathcal{P}(A^*)$ over a one-letter alphabet $A = \{a\}$ via:*

$$a^n \in L \quad \iff \quad \sigma(n) = 1, \qquad \text{for all } n \in \mathbb{N}.$$

For example, the language $L = \{a^n \mid n = 1 + 3k, k \in \mathbb{N}\}$ is represented by the state 0 in the stream automaton:



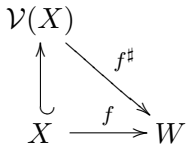
6 Linear Specifications

Equations (12) and (13) in Section 2 are examples of (finite) *linear equation systems*: the righthand side of each SDE is a linear combination of the variables on the left. In this section, we will study this type of systems in more detail.

In this section we assume A is a field. The set A^ω then becomes a vector space over A by defining scalar multiplication and vector addition element-wise, as in Table 1. We denote by $\mathcal{V}(X)$ the set of all formal linear combinations over X, i.e.,

$$\mathcal{V}(X) = \{a_1x_1 + \ldots + a_nx_n \mid a_i \in A, x_i \in X,\ \forall i : 1 \leq i \leq n\}$$

or equivalently, $\mathcal{V}(X)$ is the set of all functions from X to A with finite support. In fact, $\mathcal{V}(X)$ is itself a vector space over A by pointwise scalar multiplication and sum, and it is freely generated by X. That means X is a basis for $\mathcal{V}(X)$, and hence every linear map from $\mathcal{V}(X)$ to a vector space W is determined by its action on X. More precisely, for every function $f\colon X \to W$ there is a unique linear map $f^\sharp\colon \mathcal{V}(X) \to W$ extending f, which is defined by linearity as follows:



$$f^\sharp(a_1x_1 + \ldots + a_nx_n) = a_1f(x_1) + \ldots + a_nf(x_n).$$

We note that the linear extension $\mathrm{id}^\sharp_{A^\omega}\colon \mathcal{V}(A^\omega) \to A^\omega$ of the identity map $\mathrm{id} : A^\omega \to A^\omega$ gives the evaluation of formal linear combinations in the vector space A^ω.

6.1 Linear equation systems

A *linear equation system* over a set $X = \{x_i \mid i \in I\}$ is a collection of SDEs, one for each $x_i \in X$, of the form

$$x_i(0) \;=\; a_i, \qquad x_i' \;=\; y_i;$$

where $a_i \in A$ and $y_i \in \mathcal{V}(X)$ for all $i \in I$. In other words, a linear equation system is a map

$$e = \langle o, d\rangle \colon X \to A \times \mathcal{V}(X)$$

A *solution of* e is an assignment $h \colon X \to A^\omega$ that preserves the equations, that is, for all $x \in X = \{x_1, \ldots, x_n\}$, if $d(x) = a_1 x_1 + \ldots + a_n x_n$, then

$$h(x)(0) = o(x) \quad \text{and} \quad h(x)' = a_1 \cdot h(x_1) + \cdots + a_n \cdot h(x_n)$$

A linear equation system can also be viewed as a stream definition for the signature Σ which contains each $x \in X$ as a nullary operation together with scalar multiplication and sum. The set of closed Σ-terms is generated by the following grammar:

$$t ::= x \in X \mid a \cdot t \mid t + t, \qquad a \in A. \tag{38}$$

Note that in this perspective, an element $x \in X$ is viewed as a stream *constant*, rather than a stream *variable*, so in particular, $X \subseteq T_\Sigma(Y)$ for any set Y. The corresponding stream definition $\mathcal{D}$ consists of the SDEs defining scalar multiplication and sum, together with the SDEs from the linear equation system e. From the syntactic method (Theorem 4.8), it follows that every linear equation system e has a unique solution.

In the remainder of this section, we will characterize the solutions of finite linear equation systems in two ways. The first is in terms of rational streams, and we give a method for computing rational expressions for solutions. The second is in terms of coinduction for automata over vector spaces.

6.2 Matrix solution method

In this section we will show that solutions of finite linear equation systems are rational streams, and give a matrix-based method for computing these solutions. Recall (from [53]) that a stream $\sigma \in A^\omega$ is *rational* if it is of the form

$$\sigma = \frac{a_0 + (a_1 \times \mathsf{X}) + (a_2 \times \mathsf{X}^2) + \cdots + (a_n \times \mathsf{X}^n)}{b_0 + (b_1 \times \mathsf{X}) + (b_2 \times \mathsf{X}^2) + \cdots + (b_m \times \mathsf{X}^m)}$$

for $n, m \in \mathbb{N}$ and $a_i, b_j \in A$ and with $b_0 \neq 0$.

First, we will identify the relevant algebraic structure in which we can do matrix manipulations.

As mentioned in section 3.2, when A is a commutative ring (so, in particular, when A is a field), the stream calculus operations turn A^ω into a commutative ring $\mathbb{A} = (A^\omega, +, -, \times, [0], [1])$. For any ring R, the set $\mathsf{M}_n(R)$ of n-by-n matrices over R is again a ring under matrix addition and matrix multiplication, and when R is commutative then $\mathsf{M}_n(R)$ is an *associative R-algebra*, which means that it also has a scalar multiplication (with elements from R) which is compatible with the ring structure, that is, for all $r \in R$ and $M, N \in \mathsf{M}_n(R)$, $r \cdot (MN) = (r \cdot M)N = M(r \cdot N)$. This scalar multiplication $r \cdot M$ is defined by multiplying

each entry of M by r, that is, $(r \cdot M)_{i,j} = r \times M_{i,j}$. We refer to [39] for further results on matrix rings.

For a linear equation system with n variables, we will consider the associative A^ω-algebra $\mathsf{M}_n(A^\omega)$, and we will denote both matrix multiplication and scalar multiplication by $\cdot$. The context should make clear which operation is intended. The $\cdot$ notation is used to distinguish the operations from the multiplication in the underlying ring of stream calculus. In order to keep notation simple, we describe the matrix solution method for two variables, but it is straightforward to generalise it to n variables.

A linear equation system with two variables

$$\begin{array}{ll} x_1' = m_{11}x_1 + m_{12}x_2 & x_1(0) = n_1 \\ x_2' = m_{21}x_1 + m_{22}x_2 & x_2(0) = n_2 \end{array} \tag{39}$$

can be written in matrix form as

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}' = M \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}(0) = N$$

where derivative and initial value are taken element-wise and where

$$M = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} \qquad N = \begin{pmatrix} n_1 \\ n_2 \end{pmatrix}$$

Here m_{ij} denotes the stream $[m_{ij}] = (m_{ij}, 0, 0, 0, \ldots)$ (recall that $a \cdot \sigma = [a] \times \sigma$ for all streams σ and all $a \in A$).

By applying the fundamental theorem of stream calculus to both stream variables, we find that

$$\begin{aligned} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &= \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}(0) + \mathsf{X} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}' \\ &= N + \mathsf{X} \cdot M \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \end{aligned}$$

(Note that $\mathsf{X} \in A^\omega$, with $\mathsf{X} = (0, 1, 0, 0, 0, \ldots)$ is a scalar stream.) This is in $\mathsf{M}_n(A^\omega)$ equivalent to

$$(I - (\mathsf{X} \cdot M)) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = N$$

where I is the identity matrix. The solution to (39) can now be obtained as:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = (I - (\mathsf{X} \cdot M))^{-1} \cdot N \tag{40}$$

We should, of course, first convince ourselves that the inverse of the matrix $I - (\mathsf{X} \cdot M)$ always exists. In general, an element of a matrix ring $\mathsf{M}_n(R)$ (over a commutative ring R) is invertible if its determinant has a multiplicative inverse in R. Hence $I - (\mathsf{X} \cdot M)$ has an inverse in $\mathsf{M}_2(A^\omega)$ if its determinant is a stream whose initial value is non-zero. The matrix $I - (\mathsf{X} \cdot M)$ looks as follows

$$I - (\mathsf{X} \cdot M) = \begin{pmatrix} 1 - (\mathsf{X} \times m_{11}) & -(\mathsf{X} \times m_{12}) \\ -(\mathsf{X} \times m_{21}) & 1 - (\mathsf{X} \times m_{22}) \end{pmatrix}$$

From the definitions of sum and convolution product it follows that the initial value of the determinant equals the determinant of the matrix of initial values which in our case is

$$\det(I-(\mathsf{X}\cdot M))(0) \quad = \quad \begin{pmatrix} 1-(0\cdot m_{11}) & -(0\cdot m_{12}) \\ -(0\cdot m_{21}) & 1-(0\cdot m_{22}) \end{pmatrix} \quad = \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Hence the determinant of $I-(\mathsf{X}\cdot M)$ will always have initial value equal to 1, and consequently $(I-(\mathsf{X}\cdot M))^{-1}$ exists and can be computed using the standard linear algebra technique where the inverse of a matrix is obtained by performing elementary row operations on the identity matrix. These row operations consist of multiplying or dividing by a rational stream, and adding rows, hence if an invertible matrix has rational streams as entries, then so does its inverse. (Alternatively, this also follows from Cramer's rule.) It is easy to see that this argument carries over to higher dimensions. We have proved one direction of the following proposition.

Proposition 6.1 *A stream σ is a solution to a finite linear equation system if and only if σ is rational.*

Proof. If σ is a solution to a linear system of equations, then by the argument above this proposition, we find that σ is a linear combination of rational streams, hence itself rational. For the converse direction, if $\sigma \in A^\omega$ is rational, there exists a $d \in \mathbb{N}$ such that the d-th derivative $\sigma^{(d)}$ is a linear combination of $\sigma^{(0)}, \ldots, \sigma^{(d-1)}$. (The value d is bounded in terms of the degree of ρ and τ where $\sigma = \rho/\tau$.) Hence $\sigma^{(d)} = \sum_{i=0}^{d-1} a_i \cdot \sigma^{(i)}$ for some $a_i \in A$, $i < d$. It follows that σ is the solution for x_0 in the following d-dimensional linear equation system:

$$\begin{array}{rcl@{\qquad}rcl} x_0' &=& x_1 & x_0(0) &=& \sigma(0) \\ x_1' &=& x_2 & x_1(0) &=& \sigma(1) \\ \vdots & & \vdots & \vdots & & \vdots \\ x_{d-2}' &=& x_{d-1} & x_{d-2}(0) &=& \sigma(d-2) \\ x_{d-1}' &=& a_0x_0 + \cdots + a_{d-1}x_{d-1} & x_{d-1}(0) &=& \sigma(d-1) \end{array}$$

See also [58, Thm.5.3, Thm.5.4] for a more general proof using the vector space structure of A^ω. QED

We illustrate the matrix solution method with an example.

Example 6.2 *The Fibonacci example from* (11)

$$\sigma(0) = 0, \;\; \sigma'(0) = 1, \qquad \sigma'' = \sigma' + \sigma$$

corresponds to the linear equation system (with $x_1 = \sigma, x_2 = \sigma'$)

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}' = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}(0) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

whose solution is given by instantiating (40)*:*

$$\begin{aligned}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &= \begin{pmatrix} 1 & -\mathsf{X} \\ -\mathsf{X} & 1-\mathsf{X} \end{pmatrix}^{-1} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} \frac{1-\mathsf{X}}{1-\mathsf{X}-\mathsf{X}^2} & \frac{\mathsf{X}}{1-\mathsf{X}-\mathsf{X}^2} \\ \frac{\mathsf{X}}{1-\mathsf{X}-\mathsf{X}^2} & \frac{1}{1-\mathsf{X}-\mathsf{X}^2} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} \frac{\mathsf{X}}{1-\mathsf{X}-\mathsf{X}^2} \\ \frac{1}{1-\mathsf{X}-\mathsf{X}^2} \end{pmatrix}\end{aligned}$$

Hence the solution for σ $(= x_1)$ *is the rational stream*

$$\sigma = \frac{\mathsf{X}}{1-\mathsf{X}-\mathsf{X}^2} \tag{41}$$

By computing successive initial value and derivatives using the rational expression for σ*, we find again the Fibonacci sequence:*

$$\sigma = (0, 1, 1, 2, 3, 5, 8, 13, \ldots)$$

We end this section with some further examples of linear equation systems that define some more and some less familiar rational streams.

Example 6.3 (Naturals) *Take* $A = \mathbb{R}$*. We already saw in section 2 that the solution for* σ *in the following linear equation system is the stream of natural numbers* $\sigma = (1, 2, 3, 4, \ldots)$*:*

$$\begin{array}{rclcrcl} \sigma(0) &=& 1, & & \sigma' &=& \sigma + \tau \\ \tau(0) &=& 1, & & \tau' &=& \tau \end{array}$$

Applying the matrix solution method, we find the rational expression

$$\sigma = \frac{1}{(1-\mathsf{X})^2}$$

Example 6.4 (Powers) *Take* $A = \mathbb{R}$*. For any* $a \in \mathbb{R}$*, the linear equation*

$$\sigma(0) = 1, \quad \sigma' = a \cdot \sigma$$

has as its solution $\sigma = (1, a, a^2, a^3, a^4, \ldots)$ *with rational expression*

$$\sigma = \frac{1}{1-(a \times \mathsf{X})}$$

Example 6.5 (Alternating) *The second-order stream differential equation*

$$\sigma(0) = 0,\ \sigma'(0) = 1, \quad \sigma'' = -\sigma$$

can be written as a linear equation system

$$\begin{array}{rclcrcl} \sigma(0) &=& 0, & & \sigma' &=& \tau \\ \tau(0) &=& 1, & & \tau' &=& -\sigma \end{array}$$

The solution for σ is $\sigma = (0, 1, 0, -1, 0, 1, 0, -1, \ldots)$ with rational expression

$$\sigma = \frac{\mathsf{X}}{1 + \mathsf{X}^2}$$

Note that σ is actually eventually periodic, and could be defined by a simple equation system with four variables.

Example 6.6 (n-th powers) *For $n \in \mathbb{N}$, consider the stream $\mathsf{nats}^{\langle n \rangle} = (1, 2^n, 3^n, 4^n, \ldots)$ of n-th powers of the naturals. The stream $\mathsf{nats}^{\langle i \rangle}$ is the solution for σ_i, $i = 1, \ldots, n$, in the linear equation system in n variables defined by:*

$$\begin{array}{rclcl} \sigma_n(0) & = & 1, & \quad \sigma_n' & = & \sigma_0 + n \times \left(\sum_{i=1}^{n-1} \sigma_i \right) + \sigma_n \\ & & & & = & \sigma_0 + n\sigma_1 + \cdots + n\sigma_{n-1} + \sigma_n \end{array}$$

For example, for $n = 3$, the linear system is:

$$\begin{array}{rclcl} \sigma_0(0) & = & 1, & \quad \sigma_0' & = & \sigma_0 \\ \sigma_1(0) & = & 1, & \quad \sigma_1' & = & \sigma_0 + \sigma_1 \\ \sigma_2(0) & = & 1, & \quad \sigma_2' & = & \sigma_0 + 2\sigma_1 + \sigma_2 \\ \sigma_3(0) & = & 1, & \quad \sigma_3' & = & \sigma_0 + 3\sigma_1 + 3\sigma_2 + \sigma_3 \end{array}$$

The rational expression for $\mathsf{nats}^{\langle 3 \rangle}$ can be computed using the fundamental theorem (Theorem 3.3):

$$\begin{array}{rcll} \sigma_0 & = & 1 + X \times \sigma_0 & \Rightarrow \quad \sigma_0 = \dfrac{1}{1 - \mathsf{X}} \\ \sigma_1 & = & 1 + X \times (\sigma_1 + \sigma_0) & \\ & = & 1 + X \times (\sigma_1 + \frac{1}{1-\mathsf{X}}) & \Rightarrow \quad \sigma_1 = \dfrac{1}{(1 - \mathsf{X})^2} \\ \sigma_2 & = & 1 + X \times (\sigma_2 + 2\sigma_1 + \sigma_0) & \\ & = & 1 + X \times (\sigma_2 + \frac{2}{(1-\mathsf{X})^2} + \frac{1}{1-\mathsf{X}}) & \Rightarrow \quad \sigma_2 = \dfrac{1 + \mathsf{X}}{(1 - \mathsf{X})^3} \\ \sigma_3 & = & 1 + X \times (\sigma_3 + 3\sigma_2 + 3\sigma_1 + \sigma_0) & \\ & = & 1 + X \times (\sigma_3 + \frac{3(1+\mathsf{X})}{(1-\mathsf{X})^3} + \frac{3}{(1-\mathsf{X})^2} + \frac{1}{1-\mathsf{X}}) & \Rightarrow \quad \sigma_3 = \dfrac{1 + 4\mathsf{X} + \mathsf{X}^2}{(1 - \mathsf{X})^4} \end{array}$$

As a consequence,

$$\begin{array}{rcll} \mathsf{nats}^{\langle 0 \rangle} & = & \dfrac{1}{(1 - \mathsf{X})} & = \mathsf{ones} \\ \mathsf{nats}^{\langle 1 \rangle} & = & \dfrac{1}{(1 - \mathsf{X})^2} & = \mathsf{nats} \\ \mathsf{nats}^{\langle 2 \rangle} & = & \dfrac{1 + \mathsf{X}}{(1 - \mathsf{X})^3} & \\ \mathsf{nats}^{\langle 3 \rangle} & = & \dfrac{1 + 4\mathsf{X} + \mathsf{X}^2}{(1 - \mathsf{X})^4} & \end{array}$$

For an alternative method of computing these rational expressions and a general formula for $\mathsf{nats}^{\langle n \rangle}$*, see [48]. We also note that, in general,*

$$\mathsf{nats}^{\langle n \rangle} = \frac{A_n}{(1 - \mathsf{X})^n}$$

where A_n *is the* n*-th Eulerian polynomial.*

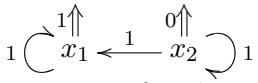
6.3 Linear stream automata

Rational streams provide an algebraic characterisation of the solutions to finite linear equation systems. In this subsection, we will give a semantic characterisation in terms of stream automata over vector spaces. Linear equation systems can be seen as specifications of *(weighted) stream automata*, cf. [10, 55, 53]. In this view, the first component of a map $\langle o, d \rangle \colon X \to A \times \mathcal{V}(X)$ assigns output weights to states, and the second component defines d an A-weighted transition structure in which state x goes to state y with weight $a \in A$ iff $d(x)(y) = a$. (Recall that $d(x)$ is a function from X to A with finite support.) Solutions to a linear equation system will now be obtained by coinduction, by showing that weighted stream automata can be viewed as (ordinary) stream automata over vector spaces.

To illustrate the construction, consider the two-dimensional linear equation system from (13), repeated here:

$$\begin{array}{rclcrcl} x_1(0) & = & 1, & & x_1' & = & x_1 \\ x_2(0) & = & 0, & & x_2' & = & x_1 + x_2 \end{array} \tag{42}$$

It corresponds to the following weighted stream automaton (output is indicated on the small double arrows):



Let us try to construct a stream automaton for the solution of x_2 by inductively applying (42) and the definition of $+$ (cf. (3)):

$$\begin{array}{rcll} x_2 & \xrightarrow{0} & x_1 + x_2 & \\ & \xrightarrow{1} & x_1 + (x_1 + x_2) & = 2 \cdot x_1 + x_2 \\ & \xrightarrow{2} & x_1 + (x_1 + (x_1 + x_2)) & = 3 \cdot x_1 + x_2 \\ & \xrightarrow{3} & \ldots & \end{array}$$

We notice two things: First, the stream behaviour of x_2 indeed consists of the sequence of natural numbers $\mathsf{nats} = (0, 1, 2, 3, 4, 5, \ldots)$. Second, the states of this stream automaton are not stream variables, but linear combinations of the stream variables x_1 and x_2.

Remark 6.7 *The above example also shows that for streams over the field* $A = \mathbb{R}$*, if the coefficients of the linear system are integers, then the solutions will be streams of integers, since all initial values will be computed using only multiplication and addition of integers.*

The above example motivates the following definition.

Definition 6.8 *Linear stream automaton A linear stream automaton is a map* $\langle o, d \rangle \colon V \to A \times V$ *where* V *is a vector space over* A*, and* $o \colon V \to A$ *and* $d \colon V \to V$ *are linear maps. Note that the pairing* $\langle o, d \rangle$ *is again linear. A homomorphism of linear stream automata is a map betwen the state vector spaces which is both linear and a homomorphism of stream automata.*

The following lemma is needed to show how solutions are obtained via coinduction for linear stream automata.

Lemma 6.9 *We have:*

1. *A linear equation system* $e\colon X \to A \times \mathcal{V}(X)$ *corresponds to a linear stream automaton* $e^\sharp\colon \mathcal{V}(X) \to A \times \mathcal{V}(X)$
2. *The final stream automaton is also a final linear stream automaton.*

Proof. Item 1: Since A is a vector space over itself, $A \times \mathcal{V}(X)$ is a (product) vector space, and we obtain $e^\sharp\colon \mathcal{V}(X) \to A \times \mathcal{V}(X)$ as the free extension of e. Note that $e^\sharp = \langle o^\sharp, d^\sharp\rangle$.

Item 2: The initial value and derivative maps are linear:

$$\begin{array}{rcl}(a\cdot\sigma + b\cdot\tau)(0) &=& a\cdot\sigma(0) + b\cdot\tau(0)\\ (a\cdot\sigma + b\cdot\tau)' &=& a\cdot\sigma' + b\cdot\tau'\end{array}$$

Hence $\langle A^\omega, \zeta\rangle$ is a linear stream automaton. Moreover, for any linear stream automaton $\langle o, d\rangle\colon V \to A \times V$, the final map $[\![-]\!]$ of the underlying (set-based) stream automata is linear. For all $v, w \in V$; $a, b \in A$ and $n \in \mathbb{N}$:

$$\begin{array}{rcll}[\![a\cdot v + b\cdot w]\!](n) &=& o(d^n(a\cdot v + b\cdot w))\\ &=& a\cdot o(d^n(v)) + b\cdot o(d^n(w)) & \text{(by linearity of } o \text{ and } d)\\ &=& a\cdot [\![v]\!](n) + b\cdot [\![w]\!](n)\end{array}$$

Hence $[\![-]\!]$ is also the unique homomorphism of linear stream automata into $\langle A^\omega, \zeta\rangle$. QED

Applying Lemma 6.9 and the coinduction principle for linear stream automata, we obtain for each linear equation system $\langle o, d\rangle\colon X \to A \times \mathcal{V}(X)$ a unique linear stream homomorphism $g\colon \mathcal{V}(X) \to A^\omega$, as shown in the following picture where $\eta_X\colon X \to \mathcal{V}(X)$ denotes the inclusion of the basis vectors into $\mathcal{V}(X)$:

$$\begin{array}{ccccc} X & \xrightarrow{\eta_X} & \mathcal{V}(X) & \xrightarrow{g} & A^\omega \\ {\scriptstyle\langle o,d\rangle}\downarrow & (1) & \swarrow{\scriptstyle\langle o^\sharp,d^\sharp\rangle} & (2) & \downarrow{\scriptstyle\zeta} \\ A\times\mathcal{V}(X) & & \xrightarrow{\mathrm{id}_A\times g} & & A\times A^\omega \end{array} \tag{43}$$

The composition $g \circ \eta_X = g\restriction_X\colon X \to A^\omega$ is a solution of $\langle o, d\rangle$ by the linearity of g. To see this, suppose that $X = \{x_1, \ldots, x_n\}$ and $d(x) = a_1x_1 + \ldots + a_nx_n$ for $x \in X$. We then have:

$$\begin{array}{rcl} g\restriction_X(x)' &=& g(d(x)) = g(a_1x_1 + \ldots + a_nx_n)\\ &=& a_1 g\restriction_X(x_1) + \ldots + a_n g\restriction_X(x_n)\end{array}$$

We note that the linear homomorphism $g\colon \mathcal{V}(X) \to A^\omega$ can be represented by a finite dimensional matrix with rational streams as entries, similar to the one in (40); see [58] or [10] for details.

6.4 Syntactic method versus linear coinduction

Given a linear equation system $\langle o, d\rangle \colon X \to A \times \mathcal{V}(X)$ we now have two ways of obtaining a map $X \to A^\omega$. One is the composition $X \xrightarrow{\eta_X} \mathcal{V}(X) \xrightarrow{g} A^\omega$ in (43) above. The other is obtained via the syntactic method by viewing linear combinations from $\mathcal{V}(X)$ as closed terms in $T_\Sigma(A^\omega)$, as explained in subsection 6.1. From the syntactic method, we then obtain a map $X \to A^\omega$ via the term interpretation $X \subseteq T_\Sigma(A^\omega) \xrightarrow{\overline{\alpha}} A^\omega$. We repeat here the relevant diagram for convenience:

$$\begin{array}{ccc} T_\Sigma(A^\omega) & \xrightarrow{\quad\overline{\alpha}\quad} & A^\omega \\ {\scriptstyle \langle o_{\mathcal{D}}, d_{\mathcal{D}}\rangle}\downarrow & & \downarrow{\scriptstyle \zeta} \\ A \times T_\Sigma(A^\omega) & \xrightarrow{\mathrm{id}_A \times \overline{\alpha}} & A \times A^\omega \end{array} \tag{44}$$

We now prove that the two methods lead to the same solution map $X \to A^\omega$ by showing that the following relation

$$R = \{\langle \overline{\alpha}(x), g(\eta_X(x))\rangle \mid x \in X\} \tag{45}$$

is a bisimulation-up-to scalar multiplication and sum, cf. Theorem 3.5. To this end, let $x \in X$ be arbitrary, and suppose that $d(x) = a_1 x_1 + \ldots + a_k x_k$.
Initial value:

$$\overline{\alpha}(x)(0) = o_{\mathcal{D}}(x) = o(x) = o^\sharp(\eta_X(x)) = g(\eta_X(x))(0).$$

Derivative: We have

$$\overline{\alpha}(x)' \overset{(44)}{=} \overline{\alpha}(d(x)) = \overline{\alpha}(a_1 x_1 + \ldots a_k x_k) = a_1 \overline{\alpha}(x_1) + \ldots + a_k \overline{\alpha}(x_k)$$

$$g(\eta_X(x))' \overset{(43)}{=} g(d(x)) = g(a_1 x_1 + \ldots a_k x_k) = a_1 g(x_1) + \ldots + a_k g(x_k)$$

where the last equalities in each line follow from $\overline{\alpha}$ being a T_Σ-algebra homomorphism, and g being linear, respectively. We have now shown that R is a bisimulation-up-to scalar multiplication and sum. It follows that for all $x \in X$, $\overline{\alpha}(x)$ and $g(\eta_X(x))$ are bisimilar, and hence by coinduction they are equal.

The equivalence between the two solution methods also follows from a more general result in [11] which relates specifications that use pure syntax (such as $T_\Sigma(X)$) and specifications that use an algebraic structure viewed as syntax modulo axioms (such as $\mathcal{V}(X)$ viewed as $T_\Sigma(X)$ modulo vector space axioms).

We can now give the semantic characterisation announced at the beginning of the section.

Proposition 6.10 *The following are equivalent for all streams $\sigma \in A^\omega$:*

1. *σ is the solution of a finite linear equation system.*
2. *σ generates a finite-dimensional subautomaton of the final linear stream automaton.*

Proof. For the direction $1 \Rightarrow 2$: Let σ be a solution to a finite $e \colon X \to A \times \mathcal{V}(X)$. Let $\langle Z_\sigma, \zeta_\sigma\rangle$ be the linear subautomaton generated by σ in the final linear automaton, i.e., the state space Z_σ is the subspace generated by the derivatives of σ. Since $\mathcal{V}(X)$ is finite-dimensional, so is its final image $g(\mathcal{V}(X))$, and since Z_σ is a subspace $g(\mathcal{V}(X))$, also Z_σ is finite-dimensional.

The direction $2 \Rightarrow 1$ follows by constructing a linear equation system using a similar argument as the one given in the proof of Proposition 6.1. A detailed proof can be found in [58, section 5, Thm.5.4]. QED

Remark 6.11 *In much of this section, we could have weakened our assumptions on A. As mentioned already, the matrix solution method only requires A to be a commutative ring. For the notion of linear automata, we only need a A to be a semiring, see the next section for a definition. A linear automaton would then be an automaton whose state space is a semimodule, rather than a vector space. Lemma 6.9 would still hold, hence coinduction for linear automata can be used as a solution method. An analogue of Proposition 6.10, however, would require that A is a so-called Noetherian semiring, cf. [22], see also [13], to ensure that the sub-semimodule generated by σ is finitely generated in the proof of the direction $1 \Rightarrow 2$.*

7 Context-free Specifications

We recall equation (9) (on page 5):

$$\sigma(0) = 1, \qquad \sigma' = \sigma \times \sigma$$

which defines the stream of Catalan numbers. It is neither simple nor linear, as the righthand side of the equation uses the convolution product. In the present section, we will study the class of *context-free* SDEs to which this example belongs.

In this section, we assume that A is a *commutative semiring*. A *semiring* is an algebraic structure $(A, +, \cdot, 0, 1)$ where $(A, +, 0)$ is a commutative monoid, $(A, \cdot, 1)$ is a monoid, multiplication distributes over addition, and 0 annihilates. A semiring $(A, +, \cdot, 0, 1)$ is *commutative*, if also $(A, +, 0)$ is a commutative monoid. The full axioms for commutative semirings are, for $a, b, c \in A$:

$$\begin{array}{lll} (a+b)+c = a+(b+c) & 0+a = a & a+b = b+a \\ (a \cdot b) \cdot c = a \cdot (b \cdot c) & 1 \cdot a = a & a \cdot b = b \cdot a \\ a \cdot (b+c) = a \cdot b + a \cdot c & (a+b) \cdot c = a \cdot c + b \cdot c & 0 \cdot a = a \end{array} \tag{46}$$

Examples of commutative semirings include the natural numbers $\mathbb{N}$ with the usual operations, and more generally any commutative ring such as the integers $\mathbb{Z}$. An important finite commutative semiring is the Boolean semiring $(2, \vee, \wedge, \bot, \top)$. More exotic examples include the tropical (min-plus) semiring $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$ and the max-plus semiring $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$. The semiring of languages over an alphabet K (with language concatenation as product) $(\mathcal{P}(K^*), \cup, \cdot, \emptyset, \{\epsilon\})$ is an example of a non-commutative semiring, i.e., one in which the product is not commutative.

Given a semiring A, we can define stream constants $[a]$ for $a \in A$, elementwise addition $+$ and convolution product $\times$ on A^ω using the SDEs in section 3.2. The algebraic structure $(A^\omega, +, \times, [0], [1])$ is again a semiring (cf. [53, Thm.4.1] and the inclusion $a \mapsto [a]$ is a homomorphism of semirings. We will therefore simply write a to denote the stream $[a]$. Note that the convolution product is commutative if and only if the underlying semiring multiplication $\cdot$ is commutative. For notational convenience, we will write $\tau\sigma$ instead of $\tau \times \sigma$ for all $\tau, \sigma \in A^\omega$.

7.1 Context-free equation systems

Let $\mathcal{M}(X^*) = \{a_0w_0 + \cdots + a_nw_n \mid a_i \in A, w_i \in X^*\}$ denote the set of formal linear combinations over the set X^* of finite words over X, or equivalently, the set of polynomials over (non-commuting) variables in X with coefficients in A. $\mathcal{M}(X^*)$ is again a semiring with the usual addition and multiplication of polynomials. If we take A to be the Boolean semiring then $\mathcal{M}(X^*)$ is the semiring of languages over alphabet X. We also note that $\mathcal{M}(X^*)$ contains A as a subsemiring via the inclusion $a \mapsto a\epsilon$, where ϵ denotes the empty word. Since we assume A is commutative, $\mathcal{M}(X^*)$ is a semiring generalisation of the notion of a unital associative algebra over a commutative ring.

A *context-free equation system over set* $X = \{x_i \mid i \in I\}$ is a collection of SDEs, one for each $x_i \in X$, of the form

$$x_i(0) \;=\; a_i, \qquad x_i' \;=\; y_i;$$

where $a_i \in A$ and $y_i \in \mathcal{M}(X^*)$ for all $i \in I$. In other words, a context-free equation system is a map $e = \langle o, d\rangle \colon X \to A \times \mathcal{M}(X^*)$.

As in the linear case, a *solution of* e is an assignment $h \colon X \to A^\omega$ that preserves the equations, that is, for all $x \in X$, if $d(x) = a_1w_1 + \ldots + a_nw_n$, then

$$h(x)(0) = o(x) \quad \text{and} \quad h(x)' = a_1h^*(w_1) + \cdots + a_nh^*(w_n)$$

where $h^*(x_1 \cdots x_n) = h(x_1) \times \cdots \times h(x_n)$. A stream σ *is context-free* if σ is the solution of some finite context-free equation system.

The name *context-free* comes from the fact that a finite context-free equation system $e = \langle o, d\rangle \colon X \to A \times M_A(X^*)$ corresponds to an A-weighted context-free grammar in Greibach normal form with non-terminals in X for a one-letter alphabet $L = \{\lambda\}$ as follows:

equation system		grammar rules
$o(x) = a$	iff	$x \to_a \epsilon$
$d(x)(w) = a$	iff	$x \to_a \lambda w, \quad w \in X^*$

where $x \to_a \lambda w$ denotes that x can produce λw with weight a. By taking A to be the Boolean semiring $\mathbb{2}$ and allowing an arbitrary alphabet L, a context-free grammar in Greibach normal form is a system of type $X \to \mathbb{2} \times T_\Sigma(X)^L$.

7.2 Solution methods

As in the linear case (see section 6) we have two methods of solving a context-free equation system e. One uses a "term view" of e, as we explain now. Consider the *polynomial signature* Σ, which contains a stream constant for each $a \in A$, and binary symbols $+$ and $\times$. The set $T_\Sigma(X)$ of all Σ-terms over a set X is generated by the following grammar:

$$t ::= x \in X \mid a \in A \mid t + t \mid t \times t \tag{47}$$

A context-free equation system over X can now be seen as a map $e \colon X \to A \times T_\Sigma(X)$ which together with the SDEs defining the polynomial Σ-operations $a \in A, +, \times$ can be viewed as one big stream definition $\mathcal{D}$ for the signature $\bar{\Sigma} = \Sigma \cup X$ where elements from X are viewed as constants. From the syntactic method, we obtain a solution map $X \subseteq T_{\bar{\Sigma}}(A^\omega) \xrightarrow{\overline{\alpha}} A^\omega$.

Also similar to the linear case, we can construct from $e\colon X \to A \times \mathcal{M}(X^*)$ a stream automaton with state space $\mathcal{M}(X^*)$ $e^\flat\colon \mathcal{M}(X^*) \to A \times \mathcal{M}(X^*)$, and apply coinduction to obtain a solution $X \to T_\Sigma(X) \xrightarrow{g} A^\omega$ from:

$$\begin{array}{ccccc} X & \xrightarrow{\eta_X} & \mathcal{M}(X^*) & \xrightarrow{g} & A^\omega \\ {\scriptstyle e}\downarrow & \swarrow{\scriptstyle e^\flat} & & & \downarrow{\scriptstyle \zeta} \\ A \times \mathcal{M}(X^*) & & \xrightarrow{\mathrm{id}_A \times g} & & A \times A^\omega \end{array} \tag{48}$$

where this time $\eta_X\colon X \to \mathcal{M}(X^*)$ denotes the inclusion of variables as polynomials. We refer to [12, 66] for details. As in the linear case, one can show that the two solutions are equal using a bisimulation-up-to polynomial operations, cf. section 6.4.

Remark 7.1 *As the above suggests, there is a more general picture that includes linear and context-free systems as special cases. This picture uses the categorical notions of monads and bialgebras (which are coalgebras on the category of algebras for a monad). In particular, the constructions $\mathcal{V}(-)$, $\mathcal{M}((-)^*)$ and $T_\Sigma(-)$ are (functor parts of) monads. We discuss this abstract approach in section 9.*

In section 6.2 we saw that solutions to linear equation systems are definable in stream calculus as the rational streams. Such closed forms for context-free streams are not generally available, but for some "quadratic" systems, it is possible to describe solutions in stream calculus using the square root operator, which is defined by (cf. [55, section 7]):

$$\sqrt{\sigma}(0) = \sqrt{\sigma(0)} \qquad (\sqrt{\sigma})' = \frac{\sigma'}{\sqrt{\sigma(0)}+\sqrt{\sigma}} \tag{49}$$

Example 7.2 (Catalan numbers) *Let $A = \mathbb{N}$ be the semiring of natural numbers. The context-free SDE from equation* (9)

$$\sigma(0) = 1, \qquad \sigma' = \sigma \times \sigma$$

defines the sequence $\sigma = (1, 1, 2, 5, 14, 42, 132, 429, 1430, \ldots)$ of Catalan numbers*, cf. [12]. In [55, p. 117-118] it is shown that the Catalan numbers are defined by the following stream calculus expression:*

$$\frac{2}{1 + \sqrt{1 - 4\mathsf{X}}}$$

Example 7.3 (Schröder numbers) *The stream differential equation*

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

has as its solution the sequence $\sigma = (1, 2, 6, 22, 90, 394, 1806, 8558, 41586, \ldots)$ of (large) Schröder numbers *(sequence A006318 in [1]), see also [66]. For $n \in \mathbb{N}$, $\sigma(n)$ is the number of paths in the $n \times n$ grid from $(0, 0)$ to (n, n) that use only single steps going right, up or diagonally right-up, and which do not go above the diagonal.*

Example 7.4 (Thue-Morse) *This example is a variation on a similar example in [12]. Let $A = \mathbb{F}_2$, the finite field $\{0,1\}$ where $1+1=0$. The following context-free system of equations*

$$\begin{array}{rclcrcl} \tau(0) & = & 0, & & \tau' & = & (\mu \times \mu) + (\mathsf{X} \times \sigma \times \sigma), \\ \sigma(0) & = & 1, & & \sigma' & = & (\sigma \times \sigma) + (\mathsf{X} \times \nu \times \nu), \\ \mu(0) & = & 1, & & \mu' & = & (\tau \times \tau) + (\mathsf{X} \times \nu \times \nu), \\ \nu(0) & = & 0, & & \nu' & = & (\nu \times \nu) + (\mathsf{X} \times \sigma \times \sigma). \end{array}$$

defines the so-called Thue-Morse sequence

$$\tau = (0,1,1,0,1,0,0,1,\ldots)$$

which, in the world of automatic sequences [4], is typically defined by means of a finite (Moore) automaton. We return to automatic sequences in section 8. Note that we could include the definition of X in the system above by adding the equations:

$$\begin{array}{rclcrcl} \mathsf{X}(0) & = & 0, & & \mathsf{X}' & = & [1] \\ [1](0) & = & 1, & & [1]' & = & [0] \\ [0](0) & = & 0, & & [0]' & = & [0] \end{array}$$

Example 7.5 *This example is taken from [54], and is not actually context-free since it uses the shuffle product $\otimes$ –rather than the convolution product – which is defined by the following SDE:*

$$(\sigma \otimes \tau)(0) = \sigma(0) \cdot \tau(0), \quad (\sigma \otimes \tau)' = (\sigma' \otimes \tau) + (\sigma \otimes \tau') \tag{50}$$

But observing that $(A^\omega, +, \otimes, [0], [1])$ also forms a semiring, it can be viewed as context-free with respect to this structure. Let $A = \mathbb{N}$, and consider the SDE

$$\sigma' = 1 + (\sigma \otimes \sigma), \quad \sigma(0) = 1$$

Its solution is the stream

$$\sigma = (1,2,4,16,80,512,3904,34816,354560,\ldots)$$

which is the sequence A000831 in [1]. The stream σ does not have a closed form, but it can be described in stream calculus by a continued fraction (cf. [52, section 17]):

$$\cfrac{\mathsf{X}}{1 - \cfrac{1 \cdot 2 \cdot \mathsf{X}^2}{1 - \cfrac{2 \cdot 3 \cdot \mathsf{X}^2}{1 - \cfrac{3 \cdot 4 \cdot \mathsf{X}^2}{\ddots}}}}$$

8 Non-standard Specifications

All stream definitions that we discussed so far make use of the same concrete, "canonical" representation of streams: a stream of elements of A consists of a first element $\sigma(0) \in A$ (the "head") followed by another stream $\sigma' \in A^\omega$ (the "tail"). There are, however, many other possible stream representations and each of these different, "non-standard" representations yield new ways of defining streams and stream functions. We are now going to discuss a few of these alternative stream representations and the resulting non-standard stream specifications.

8.1 Stream representations

Let us start by explaining what we mean by a stream representation: A representation for streams over some set A is a collection of functions that can be combined in order to turn the set A^ω into a final stream automaton (possibly of a "non-standard" type; for example, we are going to encounter stream representations that require automata in which states have two instead of one successor). This intuition has been made more precise in [37] where the corresponding, slightly more general notion is called a complete set of cooperations. Here we confine ourselves to listing a few examples.

Example 8.1

1. *We can supply the set A^ω of streams over a field A with the following structure. For $\sigma \in A^\omega$ we define*

$$\Delta\sigma = (\sigma(1) - \sigma(0),\, \sigma(2) - \sigma(1),\, \sigma(3) - \sigma(2),\, \ldots)$$

(cf. [59, 49, 55]). The Δ-operator, which is also often referred to as the forward difference operator, can be seen as a discrete derivative operator for integer functions and provides a tool for finding recurrence relations in integer sequences (cf. e.g. [59, Section 2.5]). It is not difficult to see that A^ω together with the map

$$\langle (\_)(0),\, \Delta\rangle : A^\omega \to A \times A^\omega \qquad \sigma \mapsto \langle \sigma(0),\, \Delta\sigma\rangle$$

is a final stream automaton.

2. *Another structure on A^ω is obtained by defining*

$$\frac{d}{dX}\sigma = (\sigma(1),\, 2 \cdot \sigma(2),\, 3 \cdot \sigma(3),\, \ldots)$$

for $\sigma \in A^\omega$. Again $(A^\omega, \langle (\_)(0), \frac{d}{dX}\rangle)$ is a final stream automaton. The operator $\frac{d}{dX}$ computes the derivative of a formal power series and has been used in [49] in order to establish a connection between calculus and the theory of coalgebras.

3. *In a similar fashion lots of examples could be designed: Given a set A together with some operation $o : A \times A \to A$, we define*

$$\Delta_o\sigma = (o(\sigma(0), \sigma(1)),\, o(\sigma(1), \sigma(2)),\, o(\sigma(2), \sigma(3)),\, \ldots)$$

and we can see that A^ω together with the map $\langle (\_)(0),\, \Delta_o\rangle : A^\omega \to A \times A^\omega$ is a final stream automaton provided that for any $a \in A$ the map $\lambda b.o(a, b)$ has an inverse.

But the examples for non-standard stream representations are not limited to standard stream automata as the following two interesting examples show. In order to formulate the example we need a different notion of stream automaton that we call 2-stream automaton.

Definition 8.2 *A 2-stream automaton is a set Q (of states) together with a function $\langle o, d_0, d_1\rangle : Q \to A \times Q \times Q$. A morphism between two 2-stream automata $\langle o, d_0, d_1\rangle : Q \to A \times Q \times Q$ and $\langle p, e_0, e_1\rangle : P \to A \times P \times P$ is a function $f : Q \to P$ such that $p(f(q)) = o(q)$ and $e_i(f(q)) = f(d_i(q))$ for $i = 0, 1$ and for all $q \in Q$.*

We are now ready to describe two ways of representing the set of streams as a final 2-stream automaton. The representations use the stream operations $\mathsf{even}\colon A^\omega \to A^\omega$ (which we already encountered in section 4.4), $\mathsf{odd}\colon A^\omega \to A^\omega$ and the tail of even, called $\mathsf{Even}\colon A^\omega \to A^\omega$:

$$\begin{aligned} \mathsf{odd}(\sigma) &:= (\sigma(1), \sigma(3), \dots) \\ \mathsf{even}(\sigma) &:= (\sigma(0), \sigma(2), \sigma(4), \dots) \\ \mathsf{Even}(\sigma) &:= (\sigma(2), \sigma(4), \dots) \end{aligned}$$

Example 8.3

1. *The 2-stream automaton with state set A^ω and function $\langle (\_)(0), \mathsf{odd}, \mathsf{Even}\rangle : A^\omega \to A \times A^\omega \times A^\omega$ is final among all 2-stream automata.*

2. *The set A^ω together with the function*

$$\langle (\_)(0), \mathsf{even}, \mathsf{odd}\rangle : A^\omega \to A \times A^\omega \times A^\omega$$

is not final among all 2-stream automata but among all zero-consistent 2-stream automata, i.e., among all 2-stream automata $(Q, \langle o, d_0, d_1\rangle)$ such that for all $q \in Q$ we have $o(d_0(q)) = o(q)$. It has been demonstrated in [37] that this slightly weaker finality property is sufficient for obtaining a syntactic stream definition format.

8.2 Simple non-standard specifications

We now turn to discussing stream specifications that use the above non-standard stream representations. The first thing to note is that for the representations in Example 8.1 we can easily define non-standard variations of the simple, linear and context-free specifications discussed earlier.

This can be done as follows: given any of the non-standard tail operations $\partial \in \{\Delta, \frac{d}{dX}, \Delta_o\}$ and a simple, linear or context-free equation system over a set $X = \{x_i \mid i \in I\}$ of variables with

$$x_i(0) = a_i \qquad \text{and} \qquad x_i' = t \qquad \text{for } i \in I,$$

we call the system

$$x_i(0) = a_i \qquad \text{and} \qquad \partial(x_i) = t \text{ for } i \in I,$$

obtained by replacing all derivatives x_i' with the non-standard derivatives $\partial(x_i)$, a simple, linear or context-free ∂-specification, respectively. As before, a solution to an ∂-simple, ∂-linear or ∂-context-free system of equations is a function $h : X \to A^\omega$ that preserves the equations.

Example 8.4 *Let $A = \mathbb{R}$ be the field of real numbers. The equations*

$$x(0) = 1, \qquad \Delta(x) = x$$

are an example of a simple Δ-specification of the stream

$$(1, 2, 4, 8, \dots).$$

Similarly, the equations

$$x(0) = 1, \qquad \frac{d}{dX}(x) = x$$

are a simple $\frac{d}{dX}$-specification of the stream

$$\left(\frac{1}{0!}, \frac{1}{1!}, \frac{1}{2!}, \frac{1}{3!}, \frac{1}{4!}, \dots\right).$$

The Δ-operator is particularly well studied in the area on finite analysis and the following well-known proposition provides a large class of examples of integer streams that can be defined using simple Δ-specifications.

Proposition 8.5 *A stream $\sigma \in \mathbb{R}^\omega$ has the property that $\Delta^d(\sigma) = (0, 0, 0, 0, \dots)$ for some $d \geq 0$ iff there exists a polynomial $\varphi(x)$ over $\mathbb{R}$ of degree $< d$ such that for all $n \geq 0$ we have $\sigma(n) = \varphi(n)$.*

Proof. In order to simplify the notation in the proof we write expressions of the form $a_0 + a_1 n + \cdots + a_d n^d$ in order to denote the stream that has as its n-th element $a_0 + a_1 n + \cdots + a_d n^d$, and expressions such as $a_{k+1}(k+1)!$ that contain no n will denote the constant stream $(a_{k+1}(k+1)!, a_{k+1}(k+1)!, a_{k+1}(k+1)!, \dots)$. Furthermore we use the easily verifiable fact that $\Delta(\sigma + \tau) = \Delta(\sigma) + \Delta(\tau)$ for all streams $\sigma, \tau \in \mathbb{R}^\omega$.

Suppose first that there exists some polynomial

$$\varphi(x) = a_0 + a_1 x + \cdots + a_d x^d, \qquad a_0, \dots, a_d \in \mathbb{R}, a_d \neq 0$$

of degree d with $\sigma(n) = \varphi(n)$ for all $n \in \omega$. We are going to prove by induction on d that $\Delta^d(\sigma) = (a_d d!, a_d d!, a_d d!, \dots)$. Consequently we obtain $\Delta^{d+1}(\sigma) = (0, 0, 0, \dots)$ as required.

Case $d = 0$. Then $\Delta^0(\sigma) = \sigma = (a_0 0!, a_0 0!, a_0 0!, \dots)$ as required.

Case $d = k + 1$. Then

$$\begin{aligned}
\Delta^{k+1}(\sigma) &= \Delta^{k+1}(a_0 + a_1 n + \cdots + a_{k+1} n^{k+1}) \\
&= \Delta^{k+1}(a_0 + a_1 n + \cdots + a_k n^k) + \Delta^{e+1}(a_{k+1} n^{k+1}) \\
&\overset{\text{I.H.}}{=} 0 + \Delta^k(a_{k+1}(n+1)^{k+1} - a_{k+1} n^{k+1}) \\
&= \Delta^k(a_{k+1}(k+1) n^k - r(n)) \\
&\qquad \text{where } r(n) \text{ is a poly of degree } < k \\
&\overset{\text{I.H.}}{=} a_{k+1}(k+1)k! + 0 = a_{k+1}(k+1)!
\end{aligned}$$

Consider now a stream σ such that $\Delta^d(\sigma) = (0, 0, 0, \dots)$ and suppose that $d \in \omega$ is the minimal such d. In case $d = 0$ there is nothing to prove. If $d > 0$ we have that $\Delta^{d-1}(\sigma) = (r, r, r, r, \dots)$ for some $r \neq 0$. Define $a := \frac{r}{(d-1)!}$. Then we put $\tau := \sigma - a n^{d-1}$ such that $\sigma = \tau + a n^{d-1}$. By the claim that we proved above this implies

$$\Delta^{d-1}(\sigma) = \Delta^{d-1}(\tau + a n^{d-1}) = \Delta^{d-1}(\tau) + a(d-1)! = \Delta^{d-1}(\tau) + r.$$

This clearly implies $\Delta^{d-1}(\tau) = (0, 0, 0, 0, \dots)$ and hence we can apply the I.H. to τ in order to obtain a polynomial $\varphi(x) = a_0 + a_1 x + \cdots + a_{d-2} x^{d-2}$ such that $\tau = a_0 + a_1 n + \cdots + a_{d-2} n^{d-2}$. This implies $\sigma = a_0 + a_1 n + \cdots + a_{d-2} n^{d-2} + a n^{d-1}$, ie, for all $n \in \omega$ we have $\sigma(n) = \psi(n)$ for some polynomial $\psi(x)$ of degree $< d$. QED

We are now going to compare finite simple/linear/context-free Δ- and $\frac{d}{dX}$-specifications to the corresponding tail-specifications of real-valued streams $\sigma \in \mathbb{R}^\omega$. It is not too difficult to see that the set of streams that have a finite simple Δ-specification and the set of streams that have a finite simple tail-specification are incomparable. This is demonstrated by the following examples:

Example 8.6

1. *Recall that a stream σ has finite simple* tail-*specification iff σ is ultimately periodic. Therefore the stream*
$$\sigma = (0, 1, 0, 1, 0, 1, \dots) \in \mathbb{Z}^\omega$$
has a finite simple tail-*specification. One can prove by induction that σ has infinitely many distinct Δ-derivatives which implies that σ does not have a finite simple Δ-specification. However, when $A = \mathbb{Z}/n/bbZ$ is a finite ring, σ is definable by a finite simple Δ-specification*[1].

2. *It follows from Proposition 8.5 that the stream*
$$\sigma = (0, 1, 4, 9, 25, 36, \dots)$$
with $\sigma(n) = n^2$ has a finite simple Δ-specification, but obviously no finite simple tail-*specification.*

Finite linear Δ-specifications define the same class of streams as their standard linear counterparts. This follows from the fact that
$$\Delta(\sigma) = \sigma' - \sigma \qquad \text{and} \qquad \sigma' = \Delta(\sigma) + \sigma.$$
Therefore any linear specification can be replaced by the equivalent Δ-specification:
$$\left.\begin{array}{l} x_i(0) = a_i \\ x_i' = t \end{array}\right\} \quad \Rightarrow \quad \left\{\begin{array}{l} x_i(0) = a_i \\ \Delta(x_i) = t - x_i \end{array}\right.$$
Vice versa, any linear Δ-specification can be easily transformed into an equivalent standard one. Analogously, context-free Δ-specifications and standard context-free specifications define precisely the same class of streams. We summarise our observations in the following proposition.

Proposition 8.7 *The set of streams $\sigma \in \mathbb{R}^\omega$ definable with finite simple* tail-*specifications and the set of streams definable with finite simple Δ-specifications are incomparable. Furthermore we have the following equivalences:*

- *Any stream $\sigma \in \mathbb{R}^\omega$ is definable with a finite linear* tail-*specification iff σ is definable with a finite linear Δ-specification.*
- *Any stream $\sigma \in \mathbb{R}^\omega$ is definable with a finite context-free* tail-*specification iff σ is definable with a finite context-free Δ-specification.*

[1] This observation is thanks to Michael Keane and Henning Basold.

When comparing $\frac{d}{dX}$-specifications with tail-specifications, the following identities are useful:

$$\frac{d}{dX}(\sigma) = \sigma' \odot \mathsf{nats} \tag{51}$$

$$\sigma' = \frac{d}{dX}(\sigma) \odot \mathsf{nats}^{-1} \tag{52}$$

where σ is an arbitrary stream in $\mathbb{R}^\omega$, where

$$\begin{aligned} \mathsf{nats} &= (1, 2, 3, 4, \dots) \\ \mathsf{nats}^{-1} &= (1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \dots) \end{aligned}$$

and where $\odot$ denotes the so-called Hadamard-product (pointwise multiplication) given by

$$\sigma \odot \tau = (\sigma(0)\tau(0), \sigma(1)\tau(1), \sigma(2)\tau(2), \dots).$$

This means that any simple tail-specification can be replaced by a simple $\frac{d}{dX}$-specification in which we are also allowed to employ $\odot$ and nats:

$$\left.\begin{array}{l} x_i(0) = a_i \\ x_i' = x_j \end{array}\right\} \quad \Rightarrow \quad \left\{\begin{array}{l} x_i(0) = a_i \\ \frac{d}{dX}(x_i) = x_j \odot \mathsf{nats} \end{array}\right.$$

Similarly any simple $\frac{d}{dX}$-specification can be replaced by a simple tail-specification in which we are allowed to use $\odot$ and nats^{-1}:

$$\left.\begin{array}{l} x_i(0) = a_i \\ \frac{d}{dX}(x_i) = x_j \end{array}\right\} \quad \Rightarrow \quad \left\{\begin{array}{l} x_i(0) = a_i \\ x_i' = x_j \odot \mathsf{nats}^{-1} \end{array}\right.$$

We use the description *simple* tail-nats^{-1}*-specification* for a simple tail-specification that may contain $\odot\mathsf{nats}^{-1}$ on the right hand side of the equation for the derivative a. Similarly, we define *simple* $\frac{d}{dX}$-nats*-specifications*. The above identities can be used in order to show that simple tail-nats^{-1}-specification and simple $\frac{d}{dX}$-nats-specifications are equi-expressive.

Note that without the extension by $\odot$, nats and nats^{-1} the simple tail- and $\frac{d}{dX}$-specifications are incomparable:

Example 8.8

1. *The stream $\sigma = (1, 1, 1, 1, \dots)$ has a simple* tail*-specification but no simple $\frac{d}{dX}$-specification. In order to see the second statement, we use (51) and (52) to compute:*

$$\begin{aligned} \frac{d}{dX}(\sigma) &= \mathsf{nats} \\ \frac{d}{dX}(\mathsf{nats}) &= \mathsf{nats} + \mathsf{nats} \odot \mathsf{nats} \end{aligned}$$

   and from here onwards it is easy to see that all the derivatives $(\frac{d}{dX})^r(\sigma)$ for $r \in \omega$ will be distinct and thus that σ has no finite simple $\frac{d}{dX}$-specification.

2. *The stream $\sigma = (1, 1, \frac{1}{2!}, \frac{1}{3!}, \dots)$ has a simple finite $\frac{d}{dX}$-specification (cf. Ex. 8.4) but obviously no simple finite* tail*-specification.*

8.3 Stream specifications for automatic sequences

We conclude this section by discussing simple stream specifications that make use of the stream representations from Example 8.3. What we are going to say can be developed for both representations in Example 8.3 but, following the existing literature on even-odd specifications (also called zip-specifications in [19]) we are going to use the representation from Example 8.3.2.

Definition 8.9 *A simple* even-odd*-stream definition over a set $X = \{x_i \mid i \in I\}$ of variables contains for every $x_i \in X$ three equations*

$$x_i(0) = a \qquad \mathsf{even}(x_i) = y_1^i \qquad \mathsf{odd}(x_i) = y_2^i$$

where $a \in A$ and $y_1^i, y_2^i \in X$ and where the equations entail that $x_i(0) = (\mathsf{even}(x_i))(0)$. This notion of entailment can be formalised using conditional equational logic as demonstrated in [37].

These even-odd-stream definitions are closely related to automatic sequences [4]. Let us first state the definition of a certain binary encoding of natural numbers and of automatic sequences.

Definition 8.10 *For $n \in \omega$ we define the* bbin*-encoding $\mathsf{bbin}(n)$ as the standard binary encoding read backwards, i.e., with the least significant bit first. For example: $\mathsf{bbin}(2) = 01$, $\mathsf{bbin}(5) = 101$, $\mathsf{bbin}(7) = 111$, etc.*

The following is one of several equivalent definitions of automatic sequences.

Definition 8.11 *A stream $\sigma \in A^\omega$ is called automatic if there exists a finite 2-stream automaton $\mathcal{Q}_\sigma = (Q, \langle o, d_0, d_1\rangle : Q \to A \times Q \times Q)$ and a state $q_\sigma \in Q$ such that for all $n \in \omega$ we have*

$$\sigma(n) = o\left(d_{\mathsf{bbin}(n)}(q_\sigma)\right),$$

where for $w \in 2^*$ the function $d_w : Q \to Q$ is inductively defined by $d_\epsilon(q) = q$ and $d_{wi}(q) = d_i(d_w(q))$. In other words, the n-th element of σ can be obtained as output from $\mathcal{Q}_\sigma$ by feeding the bbin*-encoding of n to the 2-stream automaton $\mathcal{Q}_\sigma$ starting from position q_σ.*

By finality there exists a unique 2-stream automaton morphism f_Q from any zero-consistent 2-stream automaton $\mathcal{Q}$ into the 2-stream automaton $(A^\omega, \langle (\_)(0), \mathsf{even}, \mathsf{odd}\rangle)$:

$$\begin{array}{ccc} Q & \overset{\exists! f_Q}{\dashrightarrow} & A^\omega \\ {\scriptstyle \langle o,d_0,d_1\rangle}\downarrow & & \downarrow{\scriptstyle \langle (\_)(0),\mathsf{even},\mathsf{odd}\rangle} \\ A \times Q \times Q & \overset{\mathrm{id}_A \times f_Q \times f_Q}{\dashrightarrow} & A \times A^\omega \times A^\omega \end{array}$$

Furthermore one can observe that for all $q \in Q$ and all $n \in \omega$ we have

$$f_Q(q)(n) = o(d_{\mathsf{bbin}(n)}(q)).$$

In other words one obtains a clean and concise characterisation of automatic sequences:

Theorem 8.12 *Let $\sigma \in A^\omega$ be a stream over some alphabet A. The following are equivalent*

1. *σ is automatic.*

2. *There exists a finite zero-consistent 2-stream automaton $(Q, \langle o, d_1, d_0\rangle)$ such that for some $q \in Q$ we have $f_Q(q) = \sigma$.*

3. *The sub-automaton of $(A^\omega, \langle (\_)(0), \mathsf{even}, \mathsf{odd}\rangle)$ generated by $\sigma \in A^\omega$ is finite.*

The proof of the theorem can be found in [38] and follows from basic universal coalgebra. As an almost immediate consequence of the theorem one obtains the following characterisation of the expressiveness of simple even-odd-stream specifications.

Corollary 8.13 *A stream $\sigma \in A^\omega$ has a finite simple* even-odd*-stream specification iff σ is automatic.*

Example 8.14 *As one example for a* even-odd*-specification consider the following simple specification of the Thue-Morse sequence from Example 7.4:*

$$\begin{array}{rclcrcl} \mathrm{TM}(0) & = & 0 & & N(0) & = & 1 \\ \mathsf{even}(\mathrm{TM}) & = & \mathrm{TM} & & \mathsf{even}(N) & = & N \\ \mathsf{odd}(\mathrm{TM}) & = & N & & \mathsf{odd}(N) & = & \mathrm{TM} \end{array}$$

Clearly the given equations entail that $(\mathsf{even}(\mathrm{TM}))(0) = \mathrm{TM}(0)$ and $(\mathsf{even}(N))(0) = N(0)$ as required by the definitionn of an even-odd*-specification. The unique solution for this specification maps* TM *to the Thue-Morse sequence.*

Much more on this way of looking at automatic sequences can be found in [19, 38]. More recently, the stream representation from Example 8.3.1 has been used in [25] to obtain stream specification formats for regular sequences, which are a linear generalisation of automatic sequences.

9 A General Perspective

As announced in Section 4, stream GSOS is a special instance of the categorical framework known as *abstract GSOS* (cf. [60]). Simply stated, abstract GSOS is obtained by generalising stream automata to F-coalgebras, and observing that a GSOS definition (for F-coalgebras) corresponds to a so-called distributive law which links algebraic structure with coalgebraic behaviour.

The survey paper [35] gives an excellent introduction to abstract GSOS, and includes many examples on streams. We present here a brief account of the categorical underpinnings of stream GSOS, and relate the general constructions to the concrete ones we have seen in earlier sections.

For this section, we assume some familiarity with basic categorical notions such as functor and natural transformation, cf. e.g. [45]. Throughout, let Set be the category of sets and functions.

9.1 Coalgebras for a functor

In previous sections, we focused on stream automata which are maps of the type $X \to A \times X$. We will now look at them from a more abstract point of view, namely as coalgebras [51, 31].

Coalgebra is a framework for studying state-based systems in a uniform setting. This is achieved by describing the system type by a functor F which defines the kind of transitions and observations the system can make. By varying F we obtain many known structures such as A-labelled binary trees ($FX = X \times A \times X$), deterministic automata ($FX = 2 \times X^A$), labelled transition systems ($FX = \mathcal{P}(X)^A$) to mention just a few. The advantage of viewing systems as F-coalgebras is that we obtain generic definitions of morphisms and bisimulation, and we can often prove results uniformly for many system types at once.

The general definition is as follows. Given a functor $F\colon \mathsf{Set} \to \mathsf{Set}$, an *$F$-coalgebra* is a pair $\langle X, c\rangle$ where X is a set and $c\colon X \to FX$ is a function. An *F-coalgebra morphism* from $\langle X, c\rangle$ to $\langle Y, d\rangle$ is a map $f\colon X \to Y$ such that $d \circ f = Tf \circ c$. An F-coalgebra $\langle Z, \zeta\rangle$ is *final* if for any F-coalgebra $\langle X, c\rangle$ there is a unique F-coalgebra morphism $h\colon \langle X, c\rangle \to \langle Z, \zeta\rangle$. An F-coalgebra bisimulation between $\langle X, c\rangle$ and $\langle Y, d\rangle$ is a relation $R \subseteq X \times Y$ which carries itself an F-coalgebra structure $r\colon R \to FR$ such that the projections $R \to X$ and $R \to Y$ are F-coalgebra morphisms. It is straightforward to check that stream automata are coalgebras for the functor $F = A \times (-)$ which maps a set X to $A \times X$ and a function $f\colon X \to Y$ to $\mathrm{id}_A \times f$. In particular, for this F we obtain as instances of F-coalgebra homomorphism and bisimulation precisely the notions of stream homomorphism and stream bisimulation, respectively, and the final $A \times (-)$-coalgebra is indeed the final stream automaton described in section 2.1.

9.2 Algebras for a monad

Where coalgebra gives us an abstract view on systems and behaviour, algebras for a monad give us an abstract view on algebraic theories, and compositionality.

We start by explaining how the usual notion of algebra can be understood categorically. An algebra for a signature Σ of operations $\underline{f}_i, i \in I$, with arities k_i, $i \in I$, is a map $\coprod_{i\in I} X^{k_i} \to X$ where X is the carrier and $\coprod$ denotes coproduct (or disjoint union). For example, if Σ contains a constant $\underline{c}$, a unary $\underline{f}$ and a binary $\underline{g}$, then a Σ-algebra with carrier X is a map $[c, f, g]\colon 1+X+(X\times X) \to X$ given by case distinction with components $c\colon 1 \to X$, $f\colon X \to X$ and $g\colon X \times X \to X$. A signature Σ corresponds in this way to a Set-functor $\Sigma X = \coprod_{i\in I} X^{k_i}$. In general, given a functor $G\colon \mathsf{Set} \to \mathsf{Set}$, a G-algebra is a pair $\langle X, \alpha\colon GX \to X\rangle$, and a G-algebra homomorphism from $\langle X, \alpha\rangle$ to $\langle Y, \beta\rangle$ is a map $h\colon X \to Y$ such that $h\circ\alpha = \beta\circ Gh$. A G-algebra $\langle X, \alpha\rangle$ is *initial* if for any G-algebra $\langle Y, \beta\rangle$ there is a unique G-algebra homomorphism $h\colon \langle X, \alpha\rangle \to \langle Y, \beta\rangle$.

Monads are functors with extra "monoid" structure. Formally, a *monad* is a triple $\langle T, \eta, \mu\rangle$ consisting of a Set-functor T, together with natural transformations $\eta\colon \mathrm{Id} \Rightarrow T$ (the *unit*), and $\mu\colon TT \Rightarrow T$ (the *multiplication*) such that $\mu \circ T\eta = \mathrm{id} = \mu \circ \eta_T$ and $\mu \circ \mu_T = \mu \circ T\mu$.

An *(Eilenberg-Moore) algebra for the monad* $\langle T, \eta, \mu\rangle$ is a T-algebra $\langle X, \alpha\rangle$ which respects the monad structure meaning that $\alpha \circ \eta_X = \mathrm{id}$ and $\alpha \circ \mu_X = \alpha \circ T\alpha$. An important role is played by $\langle TX, \mu_X\rangle$ which is the *free algebra for* $\langle T, \eta, \mu\rangle$. Given any T-algebra $\langle Y, \alpha\rangle$ and any function $f\colon X \to Y$, there is a unique T-algebra homomorphism $f^*\colon TX \to A$ such that $f^*(\eta(x)) = f(x)$ for all $x \in X$, given by $\alpha \circ Tf$.

In the remainder of this paper, we will only consider algebras for a monad (rather than for a plain functor), hence for the sake of brevity, we will refer to algebras for $\langle T, \eta, \mu\rangle$ simply as *T-algebras*, and sometimes refer to the monad simply as T leaving the unit and multiplication implicit.

We have already encountered several examples of monads. For a signature Σ, the mapping

T_Σ that assigns to a set X the set $T_\Sigma(X)$ of Σ-terms over X is the *free monad generated by the functor* Σ. The unit $\eta_X \colon X \to T_\Sigma(X)$ is inclusion of variables as terms, and the multiplication $\mu_X \colon T_\Sigma T_\Sigma(X) \to T_\Sigma(X)$ is the flattening of nested terms into terms. The notion of free monad generalises the notion of free algebra which was explained in section 4.1. The free monad comes with a universal property, since $T_\Sigma(X)$ is the initial algebra of the functor $Y \mapsto X + \Sigma(Y)$. We will also use the fact that there is a 1-1 correspondence between Σ-algebras $\Sigma X \to X$ and Eilenberg-Moore algebras $T_\Sigma X \to X$.

Another example of a monad is the construction $\mathcal{V}$ from section 6.3 where A is assumed to be a field. Recall that $\mathcal{V}(X)$ is the set of all formal linear combinations over X, i.e.,

$$\mathcal{V}(X) = \{a_1 x_1 + \ldots + a_n x_n \mid a_i \in A, x_i \in X,\ \forall i : 1 \leq i \leq n\}$$

First, $\mathcal{V}$ is a functor by defining $\mathcal{V}(f) \colon \mathcal{V}X \to \mathcal{V}Y$ by $\mathcal{V}(f)(\sum a_i x_i) = \sum b_j y_j$ where $b_j = \sum_{f(x_i)=y_j} a_i$. The unit $\eta_X \colon X \to \mathcal{V}X$ includes variables as the linear combinations: $x \mapsto 1x$, and the multiplication $\mu_X \colon \mathcal{V}^2 X \to \mathcal{V}X$ flattens by distributing scalars over sums as illustrated here for $a, b, c, d, e, f \in A$ and $x, y, z \in X$:

$$\mu_X(a(cx + dy) + b(ex + fz)) = (ac + be)x + dy + bfz.$$

The free $\mathcal{V}$-algebra $\langle \mathcal{V}X, \mu_X \rangle$ is the vector space with basis X.

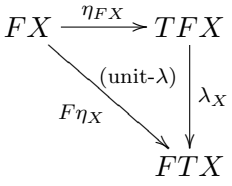
Finally, also the construction $\mathcal{M}(X^*)$ of polynomials over X with coefficients in a commutative semiring A from Section 7 is a monad with unit and multiplication defined in the expected way. For $A = \mathbb{N}$, this was shown in [30, sec. 3.4], and the proof generalises in a straightforward manner. As noted already in Section 7, the free algebra $\mathcal{M}(X^*)$ is again a semiring.

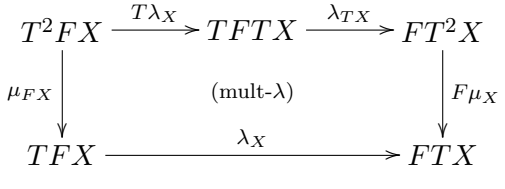
The vector space monad $\mathcal{V}$ and the polynomials monad $\mathcal{M}((-)^*)$ are examples of monads that capture equational theories. Namely, a variety of algebras defined by a signature Σ and equations/axioms E is (isomorphic to) the class of Eilenberg-Moore algebras for the monad $T_\Sigma / \equiv_E$ where $\equiv_E$ is the congruence generated by E on Σ-terms. For example, $\mathcal{V}(X)$ can be viewed as the set of "linear terms" defined in (38) quotiented with the axioms of vector spaces. Similarly, $\mathcal{M}(X^*)$ is the set of "polynomial terms" defined in (47) quotiented with the axioms of unital associative algebras over a semiring.

9.3 Bialgebras for a distributive law

The notion of a bialgebra combines coalgebraic and algebraic structure. The interaction between the two structures should be specified by a so-called *distributive law*. This definition is rather abstract at first sight, but we will later see that for a free monad T_Σ, distributive laws involving T_Σ are essentially systems of SDEs.

Given a monad T and a functor F, a *distributive law of* T *over* F is a natural transformation $\lambda \colon TF \Longrightarrow FT$ which is compatible with the monad structure, i.e., for all X the following diagrams commute:





A λ-*bialgebra* is a triple $\langle X, \alpha, \beta\rangle$ where $\alpha\colon TX \to X$ is a T-algebra and $\beta\colon X \to FX$ is an F-coalgebra which are compatible via λ, i.e., the following diagram commutes:

$$\begin{array}{ccccc} TX & \xrightarrow{\alpha} & X & \xrightarrow{\beta} & FX \\ {\scriptstyle T\beta}\downarrow & & & & \uparrow{\scriptstyle F\alpha} \\ TFX & & \xrightarrow{\lambda_X} & & FTX \end{array} \tag{53}$$

A *morphism of* λ-*bialgebras* from $\langle X_1, \alpha_1, \beta_1\rangle$ to $\langle X_2, \alpha_2, \beta_2\rangle$ is a function $f\colon X_1 \to X_2$ which is both a T-algebra morphism and an F-coalgebra morphism.

At present we are mainly interested in the case where $F = A \times (-)$ is the functor of stream automata. Fixing this F, we find that a distributive law of a monad T over F is a natural transformation whose X-component has the type $\lambda_X\colon T(A \times X) \to A \times TX$, and a λ-bialgebra has the type $TX \xrightarrow{\alpha} X \xrightarrow{\beta} A \times X$.

An important reason why distributive laws are important for solving systems of SDEs is that they induce T-algebraic structure on the final F-coalgebra, as we explain now.

Given a distributive law $\lambda\colon TF \Longrightarrow FT$, the functor F lifts to a functor F_λ on the category of T-algebras; and dually the monad T lifts to a monad T_λ on the category of F-coalgebras (cf. [7, Lem. 3.4.21]). In particular, the functor T_λ maps an F-coalgebra $\xi\colon X \to FX$ to the F-coalgebra $\lambda_X \circ T\xi\colon TX \to FTX$. (Note that $\langle TX, \mu_X, \lambda_X \circ T\xi\rangle$ is a λ-bialgebra.) Applying T_λ to the final F-coalgebra $\langle Z, \zeta\rangle$, we obtain an F-coalgebra on TZ, and hence by the finality of $\langle Z, \zeta\rangle$ there is a unique F-coalgebra morphism $\alpha\colon TZ \to Z$. For the case of stream automata, this is shown in the following diagram:

$$\begin{array}{ccccc} T(A^\omega) & \xrightarrow{T\zeta} & T(A \times A^\omega) & \xrightarrow{\lambda_{A^\omega}} & A \times T(A^\omega) \\ {\scriptstyle \alpha}\downarrow & & & & \downarrow{\scriptstyle \mathrm{id}_A \times \alpha} \\ A^\omega & & \xrightarrow{\zeta} & & A \times A^\omega \end{array} \tag{54}$$

Furthermore, it can be shown that α is an (Eilenberg-Moore) T-algebra on Z, and that $\langle Z, \alpha, \zeta\rangle$ is a final λ-bialgebra, see e.g., [7, 35] for details. In short, a distributive law $\lambda\colon TF \Longrightarrow FT$ induces a canonical T-algebra on $\langle Z, \zeta\rangle$.

Note that if $T = T_\Sigma$ is a free monad, then (54) only yields Σ-operations with no further properties such as associativity or commutativity. But if $T = T_\Sigma/{\equiv_E}$ then the $\langle Z, \zeta\rangle$ becomes a T-algebra where the equations from E hold.

This leads us to yet another reason why distributive laws and bialgebras are useful. Namely, the final F-coalgebra morphism is now also a T-algebra homomorphism. This means that the coalgebraic semantics is compositional with respect to T-algebraic structure. In particular, F-bisimilarity is a T-algebra congruence (cf. [7, Thm. 3.2.6]) which implies the soundness of *bisimulation-up-to* T-*structure*, meaning that if two states are F-bisimilar-up-to T-structure, then they are F-bisimilar. All results in section 4 are thus special instances of more general results on bialgebras.

9.4 Stream GSOS

We now show how SDEs and the syntactic method can be understood as an instance of *abstract GSOS* (cf. [60]). Abstract GSOS was developed as a general framework for structural operational semantics [3], and it is parametric in a system type F and an algebraic signature Σ. Here we fix $F = A \times (-)$ as the type of stream automata, hence the name *stream GSOS*.

9.4.1 Rule formats as natural transformations

In operational semantics, operations are usually specified by *operational rules*, rather than differential equations, but this is just a matter of notation. For example, the SDE in (3) can be equivalently formulated as a rule

$$\frac{x_0 \xrightarrow{a} y_0 \quad x_1 \xrightarrow{b} y_1}{x_0 + x_1 \xrightarrow{a+b} y_0 + y_1} \tag{55}$$

where $a, b \in \mathbb{R}$ and x_0, y_0, x_1, y_1 are read as stream variables (that correspond to $\sigma, \sigma', \tau, \tau'$ in (3)). The rule (55) can be read as defining a family ρ of functions, one for each set X, by:

$$\begin{array}{rcl} (\mathbb{R} \times X) \times (\mathbb{R} \times X) & \xrightarrow{\rho_X} & \mathbb{R} \times (X \times X) \\ \langle\langle a, y_0\rangle, \langle b, y_1\rangle\rangle & \mapsto & \langle a + b, \langle y_0, y_1\rangle\rangle \end{array} \tag{56}$$

Note how ρ_X is defined uniformly in X. Formally, ρ is a natural transformation of type

$$\rho \colon \Sigma(\mathbb{R} \times -) \Longrightarrow \mathbb{R} \times \Sigma(-) \tag{57}$$

where $\Sigma(X) = X \times X$ is the signature of a single binary operation. Such a ρ corresponds to a rule format where the derivative of an operation must be defined by a Σ-term of depth 1 over the derivatives of the arguments. This rule format can easily be extended to allow for terms of arbitrary depth by considering natural transformations of the type

$$\rho \colon \Sigma(\mathbb{R} \times -) \Longrightarrow \mathbb{R} \times T_\Sigma(-) \tag{58}$$

Natural transformations of the type $\rho \colon \Sigma(A \times (-)) \to A \times T_\Sigma(-)$ are called *stream SOS-rules*.

More generally, recall from section 4 that a stream GSOS definition for a signature Σ consists of a collection of pairs $\langle i_{\underline{f}}, d_{\underline{f}}\rangle$ one for each operation symbol $\underline{f}$ in Σ, and these pairs correspond to stream differential equations (cf. (21)). For a k-ary $\underline{f}$, this SDE can be written in the rule notation as:

$$\frac{x_1 \xrightarrow{a_1} y_1, \ \ \ldots \ \ , x_k \xrightarrow{a_k} y_k}{\underline{f}(x_1, \ldots, x_k) \xrightarrow{b} t} \tag{59}$$

where $b = i_{\underline{f}}(a_1, \ldots, a_k) \in A$ and $t = d_{\underline{f}}(a_1, \ldots, a_k)$ is a Σ-term over $x_1, \ldots, x_k, y_1, \ldots, y_k$. The dependency on $a_1, \ldots, a_k$ is usually captured by case distinction and side conditions. For example the definition of merge $\|$ in (16) (on page 6) corresponds to the rules:

$$\frac{x_1 \xrightarrow{a_1} y_1, \ x_2 \xrightarrow{a_2} y_2}{x_1 \| x_2 \xrightarrow{a_1} y_1 \| x_2} (\text{if } a_1 < a_2) \qquad \frac{x_1 \xrightarrow{a_1} y_1, \ x_2 \xrightarrow{a_2} y_2}{x_1 \| x_2 \xrightarrow{a_2} y_1 \| x_2} (\text{if } a_1 > a_2)$$

$$\frac{x_1 \xrightarrow{a_1} y_1, \ x_2 \xrightarrow{a_2} y_2}{x_1 \| x_2 \xrightarrow{a_1} y_1 \| y_2} (\text{if } a_1 = a_2)$$

The main difference with (56) and (57) is that the derivative of an operation may now be specified using not only the derivatives of the arguments (i.e. the y_i's), but also the arguments themselves (i.e. the x_i's). Here is another example of a stream GSOS rule, which defines the convolution product:

$$\frac{x_1 \xrightarrow{a_1} y_1, \ x_2 \xrightarrow{a_2} y_2}{x_1 \times x_2 \xrightarrow{0} y_1 \times x_2} (\text{if } a_1 = 0) \qquad \frac{x_1 \xrightarrow{a_1} y_1, \ x_2 \xrightarrow{a_2} y_2}{x_1 \times x_2 \xrightarrow{a_1 \cdot a_2} (y_1 \times x_2) + [a_1] \times y_2} (\text{if } a_1 \neq 0) \tag{60}$$

A collection of rules that correspond to a stream GSOS definition $\mathcal{D}$ for a signature Σ can be seen as a family of maps

$$\begin{array}{lcl} \Sigma(X \times A \times X) & \xrightarrow{\rho_X} & A \times T_\Sigma(X) \\ \underline{f}(\langle x_1, a_1, y_1\rangle, \ldots, \langle x_k, a_k, y_k\rangle) & \mapsto & \langle i_{\underline{f}}(a_1, \ldots, a_k), d_{\underline{f}}(a_1, \ldots, a_k)\rangle \end{array} \tag{61}$$

These maps constitute a natural transformation

$$\rho \colon \Sigma(- \times A \times -) \Longrightarrow A \times T_\Sigma(-) \tag{62}$$

Such a ρ is called a *stream GSOS-rule*. More generally, a GSOS-rule for a functor F and signature Σ is a natural transformation $\rho \colon \Sigma(\mathrm{Id} \times F) \Longrightarrow FT_\Sigma$.

We have thus seen how stream definitions correspond to natural transformations, and that the types of these natural transformations correspond to various rule format such as stream SOS and stream GSOS.

9.4.2 From rule formats to distributive laws

One of the aims of structural operational semantics is to identify rule formats that ensure that bisimilarity is a congruence. As we have seen in section 9.3, one way of doing so is to show that there is an appropriate distributive law. Unfortunately, distributive laws can be complicated to describe and checking the compatibility requirements with the monad can also be cumbersome.

An important insight of abstract GSOS is that SOS rules involving Σ determine distributive laws for T_Σ.

Lemma 9.1 *Let Σ be a signature functor, and $\langle T_\Sigma, \eta, \mu\rangle$ the free monad generated by Σ. For any functor F, there is a 1-1 correspondence:*

$$\begin{array}{ll} \lambda \colon T_\Sigma F \Longrightarrow FT_\Sigma & \textit{distributive law of } T_\Sigma \textit{ over } F \\ \hline \rho \colon \ \Sigma F \Longrightarrow FT_\Sigma & \textit{plain natural transformation} \end{array}$$

Proof. This is Lemma 3.4.24(i) of [7]. QED

More generally, a GSOS-rule $\rho \colon \Sigma(\mathrm{Id} \times F) \Longrightarrow FT_\Sigma$ induces a distributive law $\lambda \colon T_\Sigma(\mathrm{Id} \times F) \Longrightarrow (\mathrm{Id} \times F)T_\Sigma$ of the free monad T_Σ over the *cofree copointed functor over F* which consists of the functor $\mathrm{Id} \times F$ together with the natural (left) projection $\pi_1 \colon \mathrm{Id} \times F \Longrightarrow \mathrm{Id}$, see [41] for further details. We call such a distributive law a *GSOS law*. A GSOS law must respect not only the monad structure, but also the copointed structure meaning that $\pi_1 \circ \lambda = T_\Sigma \pi_1$.

Lemma 9.2 *Let Σ be a signature functor, and $\langle T_\Sigma, \eta, \mu\rangle$ the free monad generated by Σ. For any functor F, there is a 1-1 correspondence between GSOS-rules for Σ and distributive laws of T_Σ over the cofree copointed functor $\langle \mathrm{Id} \times F, \pi_1\rangle$.*

$$\begin{array}{ll} \lambda \colon T_\Sigma(\mathrm{Id} \times F) \Longrightarrow (\mathrm{Id} \times F)T_\Sigma & \textit{distributive law of } T_\Sigma \textit{ over } \langle \mathrm{Id} \times F, \pi_1\rangle \\ \hline \rho \colon \ \Sigma(\mathrm{Id} \times F) \Longrightarrow FT_\Sigma & \textit{plain natural transformation} \end{array}$$

Proof. This is Lemma 3.5.2(i) of [7]. QED

Instantiating the above for the stream functor $F = A \times (-)$, we have that a stream GSOS-rule $\rho\colon \Sigma((-) \times A \times (-)) \Longrightarrow A \times T_\Sigma(-)$ induces a stream GSOS law $\lambda\colon T_\Sigma((-) \times A \times (-)) \Longrightarrow T_\Sigma(-) \times (A \times T_\Sigma(-))$. For any stream automaton $\beta\colon X \to A \times X$ such a GSOS law λ yields a stream automaton structure on $T_\Sigma(X)$ by

$$T_\Sigma(X) \xrightarrow{T_\Sigma\langle \mathrm{id}_X, \beta\rangle} T_\Sigma(X \times (A \times X)) \xrightarrow{\pi_2 \circ \lambda_X} A \times T_\Sigma(X)$$

Applying this construction to the final stream automaton $\zeta\colon A^\omega \to A \times A^\omega$, we obtain precisely the stream automaton $\langle o_\mathcal{D}, d_\mathcal{D}\rangle$ from Definition 4.3, and hence a unique stream automaton homomorphism $\alpha_\lambda\colon T_\Sigma(A^\omega) \to A^\omega$ by coinduction, as shown in the following diagram:

$$\begin{array}{ccccc} T_\Sigma(A^\omega) & \xrightarrow{T_\Sigma\langle \mathrm{id}, \zeta\rangle} & T_\Sigma(A^\omega \times (A \times A^\omega)) & \xrightarrow{\lambda_{A^\omega}} & A \times T_\Sigma(A^\omega) \\ \downarrow{\scriptstyle \alpha_\lambda} & & & & \downarrow{\scriptstyle \mathrm{id}_A \times \alpha_\lambda} \\ A^\omega & & \xrightarrow[\cong]{\quad\zeta\quad} & & A \times A^\omega \end{array} \tag{63}$$

This α_λ is thus the final homomorphism $[\![-]\!]$ from Section 4.

To summarise, a collection of SDEs that together form a stream GSOS definition as described in Section 4.2, corresponds to a stream GSOS law and the stream operations are obtained via the final map α_λ.

9.4.3 Distributive laws for non-free monads

We have now seen how SDEs correspond to distributive/GSOS laws for free monads, which in turn define operations on the final coalgebra. However, the two monads $\mathcal{V}$ and $\mathcal{M}(-^*)$ relevant for linear and context-free systems are not free. Concretely, the SDEs that define scalar multiplication and addition of streams do not automatically ensure that these stream operations satisfy the axioms of vector spaces. In other words, the SDEs only induce operations, and not actual vector space structure, on the final stream automaton. However, we know that the final stream automaton is a vector space (cf. section 3.2). This can be proved directly, but it also follows from the existence of a distributive law $\lambda\colon \mathcal{V}(A \times (-)) \Longrightarrow A \times \mathcal{V}(-)$ given by (cf. [32, Thm. 10]) :

$$\lambda_X : \mathcal{V}(A \times X) \xrightarrow{\langle \mathcal{V}\pi_1, \mathcal{V}\pi_2\rangle} \mathcal{V}A \times \mathcal{V}X \xrightarrow{\beta \times \mathrm{id}_{\mathcal{V}X}} A \times \mathcal{V}X \tag{64}$$

where $\beta\colon \mathcal{V}A \to A$ is the vector space structure on the field A, and π_1, π_2 denote left and right projection, respectively. Working out the details of this λ shows that the $\mathcal{V}$-algebra (i.e. vector space structure) induced on A^ω is exactly given by the pointwise operations of scalar multiplication and addition that are also defined by the SDEs. This way of obtaining a λ easily generalises to any pointwise defined stream operations.

In order to solve context-free systems, we would like to have a distributive law for $T = \mathcal{M}((-)^*)$. Note, however, that we cannot simply replace $\mathcal{V}$ by $\mathcal{M}((-)^*)$ in (64) above, since the induced pointwise semiring structure on streams is not the one we are interested in, as the convolution product of streams A^ω is *not* the pointwise extension of the semiring product on A. Nevertheless, there is a stream GSOS law $\lambda\colon \mathcal{M}(((-) \times A \times (-))^*) \Longrightarrow \mathcal{M}((-)^*) \times A \times \mathcal{M}((-)^*)$

that induces the desired semiring structure on streams. The existence of this λ can be proved by showing that the SDEs defining sum and convolution product respect the semiring axioms in a certain sense. We refer to [11] for a detailed argument.

9.5 Solving systems of equations

We will now show how the bialgebraic picture subsumes the coinductive methods used to solve linear and context-free equation systems.

We saw in sections 6 and 7 that linear equation systems are maps of the form $e\colon X \to A \times \mathcal{V}X$, and context-free equation systems are maps of the form $e\colon X \to A \times \mathcal{M}(X^*)$. More generally, a system of equations for $T$and F is a map $e\colon X \to FTX$. If we have a distributive law $\lambda\colon TF \Longrightarrow FT$, then for every equation system $e\colon X \to FTX$, we can construct a λ-bialgebra $\langle TX, \mu_X, e_\lambda\rangle$ with free T-algebra component by taking $e_\lambda = F\mu_X \circ \lambda_{TX} \circ Te$, cf. [7, Lemma 4.3.3]. We now obtain a unique λ-bialgebra morphism g into the final λ-bialgebra, as shown here for the stream functor $F = A \times (-)$:

$$\begin{array}{ccccc} T^2(X) & & \xrightarrow{Tg} & & T(A^\omega) \\ \downarrow{\scriptstyle \mu_X} & & & & \downarrow{\scriptstyle \alpha} \\ X & \xrightarrow{\eta_X} & T(X) & \xrightarrow{g} & A^\omega \\ \downarrow{\scriptstyle e} & \swarrow{\scriptstyle e_\lambda} & & & \downarrow{\scriptstyle \zeta} \\ A \times T(X) & & \xrightarrow{\mathrm{id}_A \times g} & & A \times A^\omega \end{array} \tag{65}$$

Note that diagrams (43) for linear solutions and (48) for context-free solutions are both instances of (65); except that the free algebra part was left implicit. In general, the map $g \circ \eta_X\colon X \to A^\omega$ defined as in (65) is a (unique) stream solution to the equation system $e\colon X \to A \times T(X)$, cf. [7, Lemma 4.3.4]. We also note that the above generalises to GSOS laws $\lambda\colon T_\Sigma(\mathrm{Id} \times F) \Longrightarrow (\mathrm{Id} \times F)T_\Sigma$, but the argument is more involved, see e.g. [7, Lem. 3.5.2] and also [30, 41].

In summary, a distributive/GSOS law λ ensures the existence of unique solutions in the final F-coalgebra to equation systems of the type $e\colon X \to FTX$. If $T = T_\Sigma$ is a free monad for a signature Σ, then λ is essentially given by a collection of SDEs that define Σ-operations. If T is not free, then we cannot immediately claim the existence of a λ (and hence unique solutions) by giving a system of SDEs. However, when T encodes a variety of algebras in terms of operations Σ and equations E, (such as, for example, $\mathcal{V}$ or $\mathcal{M}((-)^*)$), then λ can often be described as a quotient of the law λ_Σ that corresponds to a system of SDEs. In this case, the solutions obtained from λ coincide with the solutions obtained from λ_Σ. These results are described in detail in [11].

Note that for $T = T_\Sigma$, there is a subtle difference between the syntactic solution method and (65). Namely, the syntactic method puts the definitions of the stream operations as well as the equation system into one big stream definition which eventually corresponds to a GSOS law, whereas the solution obtained via (65) separates the definition of operations (in λ) from the equation system itself.

9.5.1 Linear equation systems, revisited

Let A be a field. The behaviour functor is the stream functor $F = A \times (-)$, T is the vector space monad $\mathcal{V}$ described in section 9.2, and let λ be as in (64). A linear equation system is a map $e\colon X \to A \times \mathcal{V}X$, and by (65), a unique solution always exists. This solution method is essentially the same as linear coinduction. Namely, for this λ, λ-bialgebras are the same as linear automata.

In the syntactic view, we take as λ the SDES defining sum and scalar multiplication, and a linear equation system is viewed as a map $e\colon X \to A \times T_\Sigma(X)$ where Σ is the "linear signature" described before (38) (on page 22). This syntactic view does not take into account the equational theory of vector spaces. This means that, for example, the following two linear equation systems

$$\begin{array}{ll} x(0) = 0, & x' = 2x + 3y, \\ y(0) = 1, & y' = x, \end{array} \qquad\qquad \begin{array}{ll} x(0) = 0, & x' = 3y + 2x, \\ y(0) = 1, & y' = x, \end{array} \tag{66}$$

are considered the same when viewed as being of type $e\colon X \to A \times \mathcal{V}X$, but different when viewed as being of type $e\colon X \to A \times T_\Sigma(X)$. Hence, if we apply (65) to solve these two systems, it is not apriori clear that the two systems have the same stream solutions. However, since the vector space operations on streams obtained via (64) coincide with the operations defined by the SDEs, the solutions of the systems in (66) also coincide.

9.5.2 Context-free equation systems, revisited

Now we assume that A is a commutative semiring. The behaviour functor is again the stream functor $F = A \times (-)$ and T is the polynomial monad $\mathcal{M}((-)^*)$ described at the end of section 9.2. As explained in section 9.4.3, there is a GSOS law λ for $\mathcal{M}((-)^*)$. This λ is a quotient of the syntactic GSOS law λ_Σ that corresponds to the SDEs that define semiring operations, cf. [11]. Hence by (65), every context-free equation system $e\colon X \to A \times \mathcal{M}(X^*)$ has a unique stream solution. Similar considerations apply as those discussed above for linear equation systems when we take a syntactic view on context-free systems.

10 Other Specification Methods

There exist many ways of representing streams, other than by stream differential equations. Among the classical methods in mathematics are recurrence relations, generating functions and continued fractions. In computer science, weighted automata are also often used (cf. section 6.3). As a basic and instructive example, we use the stream of Fibonacci numbers

$$\phi = (0, 1, 1, 2, 3, 5, 8, 13, \ldots)$$

to quickly illustrate a number of different stream representations.

We already saw a definition of ϕ by means of a stream differential equation (cf. (11)):

$$\phi(0) = 0 \quad \phi(1) = 1 \qquad \phi'' = \phi + \phi' \tag{67}$$

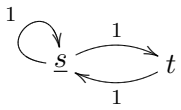
A definition of ϕ by means of a *recurrence relation* is the following:

$$\phi(0) = 0 \quad \phi(1) = 1 \qquad \phi(n+2) = \phi(n) + \phi(n+1) \tag{68}$$

The following representation is called in mathematics an (ordinary) *generating function*:

$$f(x) = \frac{x}{1 - x - x^2} \tag{69}$$

It corresponds to the rational expression $\frac{\mathsf{X}}{1-\mathsf{X}-\mathsf{X}^2}$, which we already saw in (41). The expansion of $f(x)$ into $f(x) = x + x^2 + 2x^3 + 3x^4 + 5x^5 + \cdots$ gives us the Fibonacci numbers. Finally, the value of the nth Fibonacci number can be read from this weighted automaton (where a state is underlined if its output is 1, otherwise the output is 0)



(70)

by counting the number of finite paths of length n leading from the state s back to the state s again.

All is well with this basic example. All four representations above (and still others) are well-understood, including the way to obtain one from the other (as we have seen in section 6). But things get much less clear very quickly. Consider for instance the stream of factorial numbers $\psi = (0!, 1!, 2!, 3!, \ldots)$. A recurrence relation is again easily given:

$$\psi(0) = 1 \qquad \psi(n+1) = (n+1) \cdot \psi(n) \tag{71}$$

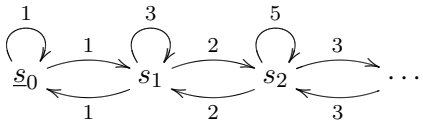
but now look at the following stream differential equation, also defining ψ:

$$\psi(0) = 1 \qquad \psi' = \psi \otimes \psi \tag{72}$$

where the righthand side uses the shuffle product (defined in (50)). It is unclear how (71) and (72) are related. Furthermore, we know of no (ordinary) generating function for ψ but then again, there is the following continued fraction:

$$\psi = \cfrac{1}{1 - x - \cfrac{1^2x^2}{1 - 3x - \cfrac{2^2x^2}{1 - 5x - \cfrac{3^2x^2}{\ddots}}}} \tag{73}$$

as well as the following representation of ψ by means of an (infinite) weighted automaton



(74)

For this example, the relation between (73) and (74) is fairly direct but, more generally, the relation between all four different representations (71)-(74) of the factorial numbers is by no means well-understood, and serves as an illustration of an interesting class of problems that need further study.

11 Related Work

We have given an overview of recent results on streams obtained via a coalgebraic perspective. In this section we give pointers to the surveyed literature, and a brief overview of some related work, which is bound to be incomplete.

Formal power series and automata theory Streams are formal power series in only one variable and as a consequence, many of the properties of streams and stream differential equations presented here are ultimately special instances of more general facts about formal power series. We mention [9] as a fundamental reference on formal power series in multiple noncommutative variables, and refer to [64] for an extensive discussion of the relationship between the coalgebraic and the classical approaches to streams and formal power series.

Streams and coalgebra The coalgebraic treatment of streams, stream differential equations and stream calculus started with [52, 53]. Section 6 on linear specifications is based on work found in the just mentioned papers, as well as further investigations into rational streams and linear systems in [58, 10]. Section 7 on context-free specifications is based on [12, 65, 66]. Previously, context-free languages were studied coalgebraically in [27], but using a different approach, see [66, sec. 1.1] for a discussion. Section 8 on non-standard specifications is based on work in [37] and for automatic sequences on [19, 38]. Further work in this direction includes [25] on k-regular sequences.

Other coalgebraic investigations into streams and stream functions include the following. Specification formats and coalgebraic semantics (as Mealy machines) for stream functions in 2-adic arithmetic have been studied in [57, 26]. Causal stream functions generalise to continuous stream functions, which have been characterised categorically in [23].

Stream circuits Linear circuits (or signal flow graphs) are another representation of streams (which we did not include in our survey). In [56] it was shown that rational streams are exactly the streams that can be defined by closed linear circuits. An axiomatisation of rational streams in a fixed point calculus was given in [46]. Recently, the semantics of open linear circuits was given a coalgebraic and algebraic characterisation in [8], which leads also to a complete axiomatisation in a calculus of commutative rings and modules.

Morphic and automatic sequences Yet another way of specifying streams which comes from the field of combinatorics on words is as a limit of a (monoid) morphism, see e.g. [42, Ch. 10] and [4, Ch. 7]. A translation between morphic definitions and coinductive definitions was given in [21, Sec. 2]. Coalgebraic characterisations of automatic and regular streams were given in [19, 38, 25]

Abstract GSOS Abstract GSOS originated as a categorical approach to structural operational semantics [3]. The seminal paper on the topic is [60], and [35] provides an introductory overview, which also contains many examples for streams. Other rich sources of general results on bialgebras and distributive laws are [6, 7, 34, 40, 41, 61]. See also [30, 32] for a bialgebraic treatment of formal languages and regular expressions, and several other examples. In [24], it is shown that a stream GSOS definition D can be transformed into a GSOS defintion C for

causal stream functions that defines the pointwise extensions of the stream operations defined by D.

Functional programming Lazy functional programming languages, such as Haskell, allow programming on streams, and leads to many interesting examples and applications [17, 29]. Here it is also of interest to find methods of ensuring that a program operating on streams (or, more generally, on codata) is well-defined. Specification formats for codata in functional languages have been studied in, e.g., [2, 5]. Functional programming on non-wellfounded structures such as stream automata was studied in [33].

Term rewriting Closely related to functional programming is the work on streams in term rewriting. In particular, the productivity of stream specifications given as term rewrite systems is studied in [18, 20, 68].

Tools Several tools exist for specifying and reasoning about streams using stream differential equations. We mention just a few. The rewriting-based tool *CIRC [44, 43, 50]* can check equivalence of stream specifications (i.e., whether they define the same stream) using circular coinduction. The tool *Streambox [69]* uses more general equational reasoning combined with circular coinduction to prove equivalence of stream specifications. The Haskell-based tool *QStream [63, 62]* provides facilities for entering stream differential equations, and exploring streams together with interfacing with the OEIS [1].

References

[1] Sloane's online encyclopedia of integer sequences. `http://oeis.org`.

[2] A. Abel and B. Pientka. Well-founded recursion with copatterns. In Morrisett and Uustalu [47], pages 185–196.

[3] L. Aceto, W.J. Fokkink, and C. Verhoef. Structural operational semantics. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 197–292. Elsevier, 2001.

[4] J.-P. Allouche and J. Shallit. *Automatic Sequences: Theory, Applications, Generalizations*. Cambridge University Press, 2003.

[5] R. Atkey and C. McBride. Productive coprogramming with guarded recursion. In Morrisett and Uustalu [47], pages 197–208.

[6] F. Bartels. Generalised coinduction. *Mathematical Structures in Computer Science*, 13:321–348, 2003.

[7] F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

[8] H. Basold, M.M. Bonsangue, H.H. Hansen, and J.J.M.M. Rutten. (Co)algebraic characterizations of signal flow graphs. In F. van Breugel, E. Kashefi, C. Palamidessi, and J. Rutten, editors, *Horizons of the Mind: A Tribute to Prakash Panangaden*, volume 8464 of *Lecture Notes in Computer Science*, pages 124–145. Springer, 2014.

[9] J. Berstel and C. Reutenauer. *Noncommutative Rational Series with Applications*. Cambridge University Press, 2011.

[10] F. Bonchi, M. Bonsangue, Boreale M., Rutten J.J.M.M., and Silva A. A coalgebraic perspective on linear weighted automata. *Information and Computation*, 211:77–105, 2012.

[11] M.M Bonsangue, H.H Hansen, A. Kurz, and Rot J. Presenting distributive laws. In R. Heckel and S. Milius, editors, *Proceedings of CALCO 2013*, Lecture Notes in Computer Science, 2013. To appear.

[12] M.M. Bonsangue, J.J.M.M. Rutten, and J. Winter. Defining context-free power series coalgebraically. In D. Pattinson and L. Schroeder, editors, *Proceedings of CMCS 2012*, volume 7399 of *Lecture Notes in Computer Science*, pages 20–39. Springer, 2012.

[13] M.M. Bonsangue, Milius S., and Silva A. Sound and complete axiomatizations of coalgebraic language equivalence. *ACM Transactions on Computational Logic*, 13, 2012.

[14] J.A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.

[15] J.H. Conway. *Regular algebra and finite machines.* Chapman and Hall, 1971.

[16] E.W. Dijkstra. Hamming's exercise in SASL. Handwritten note EWD792, University of Texas, 1981.

[17] K. Doets and J. van Eijck. *The Haskell Road to Programming.* Texts in Computing. College Publications, 2nd edition, 2012.

[18] J. Endrullis, C. Grabmayer, D. Hendriks, A. Isihara, and J.W. Klop. Productivity of stream definitions. *Theoretical Computer Science*, 411(4-5):765–782, 2012.

[19] J. Endrullis, C. Grabmayer, D. Hendriks, J.W. Klop, and L.S. Moss. Automatic sequences and zip-specifications. In N. Dershowitz, editor, *Proceedings of LICS 2012*, 2012.

[20] J. Endrullis and D. Hendriks. Lazy productivity via termination. *Theoretical Computer Science*, 412(28):3203–3225, 2011.

[21] J. Endrullis, D. Hendriks, and M. Bodin. Circular coinduction in Coq using bisimulation-up-to techniques. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Proc. 4th Int. Conf. on Interactive Theorem Proving (ITP 2013)*, volume 7998 of *Lecture Notes in Computer Science*, pages 354–369. Springer, 2013.

[22] Z. Ézik and A. Maletti. The category of simulations for weighted tree automata. *International Journal of Foundations of Computer Science (IJFCS)*, 22(8):1845–1859, 2011.

[23] N. Ghani, P. Hancock, and D. Pattinson. Representations of stream processors using nested fixed points. *Logical Methods in Computer Science*, 5(3), 2009.

[24] H.H. Hansen and B. Klin. Pointwise extensions of GSOS-defined operations. *Mathematical Structures in Computer Science*, 21:321–361, 2011.

[25] H.H. Hansen, C. Kupke, J.J.M.M. Rutten, and Winter J. A final coalgebra for k-regular sequences. In F. van Breugel, E. Kashefi, C. Palamidessi, and J. Rutten, editors, *Horizons of the Mind: A Tribute to Prakash Panangaden*, volume 8464 of *Lecture Notes in Computer Science*, pages 363–383. Springer, 2014.

[26] H.H. Hansen and J.J.M.M. Rutten. Symbolic synthesis of mealy machines from arithmetic bitstream functions. *Scientific Annals of Computer Science*, 20:97–130, 2010.

[27] I. Hasuo and B. Jacobs. Context-free languages via coalgebraic trace semantics. In J.L. Fiadeiro, N. Harman, M. Roggenbach, and J. Rutten, editors, *Proceedings of CALCO*, volume 3629 of *Lecture Notes in Computer Science*, pages 213–231. Springer, 2005.

[28] E.C.R. Hehner and R.N. Horspool. A new representation of the rational numbers for fast easy arithmetic. *SIAM Journal on Computing*, 8:124–134, 1979.

[29] R. Hinze. Concrete stream calculus: An extended study. *J. Funct. Program.*, 20(5-6):463–535, 2011.

[30] B. Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In K. Futatsugi, J.-P. Jouannaud, and J. Meseguer, editors, *Algebra, Meaning and Computation: Essays dedicated to Joseph A. Goguen on the Occasion of his 65th Birthday*, volume 4060 of *Lecture Notes in Computer Science*, pages 375–404. Springer, 2006.

[31] B. Jacobs and J.J.M.M. Rutten. An introduction to (co)algebras and (co)induction. In D. Sangiorgi and J.J.M.M. Rutten, editors, *Advanced topics in bisimulation and coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2011.

[32] Bart Jacobs. Distributive laws for the coinductive solution of recursive equations. *Inf. Comput.*, 204(4):561–587, 2006.

[33] J.-J. Jeannin, D. Kozen, and A. Silva. Language constructs for non-well-founded computation. In M. Felleisen and P. Gardner, editors, *22nd European Symposium on Programming (ESOP 2013)*, volume 7792 of *Lecture Notes in Computer Science*, pages 61–80, Rome, Italy, March 2013. Springer.

[34] B. Klin. Bialgebraic methods and modal logic in structural operational semantics. *Information and Computation*, 207(2):237–257, 2009.

[35] B. Klin. Bialgebras for structural operational semantics: An introduction. *Theoretical Computer Science*, 412:5043–5069, 2011.

[36] C. Kupke, M. Niqui, and J.J.M.M. Rutten. Stream differential equations: concrete formats for coinductive definitions. Technical Report RR-11-10, University of Oxford, 2011. To appear as a book chapter.

[37] C. Kupke and J.J.M.M. Rutten. Complete sets of cooperations. *Inf. Comput.*, 208(12):1398–1420, 2010.

[38] C. Kupke and J.J.M.M. Rutten. On the final coalgebra of automatic sequences. In R.L. Constable and A. Silva, editors, *Festschrift for Dexter Kozen*, volume 7230 of *Lecture Notes in Computer Science*. Springer, 2012. CWI Technical Report SEN-1112, 2011.

[39] S. Lang. *Algebra*. Graduate Texts in Mathematics. Springer, 2002.

[40] M. Lenisa, J. Power, and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *Electr. Notes Theor. Comput. Sci.*, 33:230–260, 2000.

[41] M. Lenisa, J. Power, and H. Watanabe. Category theory for operational semantics. *Theoretical Computer Science*, 327(1-2):135–154, 2004.

[42] M. Lothaire. *Applied Combinatorics on Words*. Cambridge University Press, 2005.

[43] D. Lucanu, E.-I.Goriac, G; Caltais, and G. Rosu. CIRC: a behavioral verification tool based on circular coinduction. In A. Kurz, M. Lenisa, and A. Tarlecki, editors, *Proceedings of CALCO*, volume 5728 of *Lecture Notes in Computer Science*, pages 433–442, 2009.

[44] D Lucanu and G. Rosu. CIRC: a circular coinductive prover. In T. Mossakowski, U. Montanari, and M. Haveraaen, editors, *Proceedings of CALCO*, volume 4624 of *Lecture Notes in Computer Science*, pages 372–378, 2007.

[45] S. MacLane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 1998.

[46] S. Milius. A sound and complete calculus for finite stream circuits. In *Proceedings of LICS*, pages 449–458. IEEE Computer Society, 2010.

[47] G. Morrisett and T. Uustalu, editors. *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, New York, NY, USA, 2013. ACM.

[48] M. Niqui and J.J.M.M. Rutten. An exercise in coinduction: Moessners theorem. Technical Report SEN-1103, Centrum Wiskunde & Informatica, 2011.

[49] D. Pavlovic and M.H. Escardó. Calculus in coinductive form. In *Proceedings of LICS 1998*, pages 408–417. IEEE Society, 1998.

[50] G. Rosu. CIRC tool webpage. URL: `http://fsl.cs.illinois.edu/index.php/Circ`.

[51] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[52] J.J.M.M. Rutten. Elements of stream calculus (an extensive exercise in coinduction). In S. Brooks and M. Mislove, editors, *Proceedings of MFPS 2001*, volume 45 of *Electronic Notes in Theoretical Computer Science*, pages 1–66. Elsevier Science Publishers, 2001.

[53] 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.

[54] J.J.M.M. Rutten. Coinductive counting with weighted automata. *Journal of Automata, Languages and Combinatorics*, 8(no. 2):319–352, 2003.

[55] J.J.M.M. Rutten. A coinductive calculus of streams. *Mathematical Structures in Computer Science*, 15:93–147, 2005.

[56] J.J.M.M. Rutten. A tutorial on coinductive stream calculus and signal flow graphs. *Theoretical Computer Science*, 343(3):443–481, 2005.

[57] J.J.M.M. Rutten. Algebraic specification and coalgebraic synthesis of Mealy machines. In *Proceedings FACS 2005*, volume 160 of *ENTCS*, pages 305–319, 2006.

[58] J.J.M.M. Rutten. Rational streams coalgebraically. *Logical Methods in Computer Science*, 3:9:1–22, 2008.

[59] N. Sloane and S. Plouffe. *The Encyclopedia of Integer Sequences*. 1995.

[60] D. Turi and G.D. Plotkin. Towards a mathemathical operational semantics. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS 1997)*, pages 280–291. IEEE Computer Society, 1997.

[61] H. Watanabe. Well-behaved translations between structural operational semantics. In L. Moss, editor, *Proceedings of CMCS 2002*, volume 65 of *Electronic Notes in Theoretical Computer Science*, pages 337–357. Elsevier, 2002.

[62] J. Winter. QStream tool gwebpage. URL: `http://homepages.cwi.nl/~winter/qstream/`.

[63] J. Winter. QStream: a suite of streams. In R. Heckel and S. Milius, editors, *Proceedings of CALCO*, volume 8089 of *Lecture Notes in Computer Science*, pages 353–358. Springer, 2013.

[64] J. Winter. *Coalgebraic Characterizations of Automata-theoretic Classes*. PhD thesis, Radboud Universiteit Nijmegen, 2014.

[65] J. Winter, M.M. Bonsangue, and J.J.M.M. Rutten. Context-free coalgebras. To appear.

[66] J. Winter, M.M. Bonsangue, and J.J.M.M. Rutten. Coalgebraic characterizations of context-free languages. *Logical Methods in Computer Science*, 9(3:14), 2013.

[67] C.K. Yuen. Hamming numbers, lazy evaluation, and eager disposal. *ACM SIGPLAN Notices*, 27(issue 8):71–75, 1992.

[68] H. Zantema. Well-definedness of streams by transformation and termination. *Logical Methods in Computer Science*, 6(3):paper 21, 2010.

[69] H. Zantema and J. Endrullis. Proving equality of streams automatically. In M. Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, pages 393–408, 2011.