A tutorial on coinductive stream calculus and signal flow graphs

J.J.M.M. Rutten*

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

Abstract

This paper presents an application of coinductive stream calculus to signal flow graphs. In comparison to existing approaches, which are usually based on Z-transforms (a discrete version of Laplace transforms) and transfer functions, the model presented in these notes is very elementary. The formal treatment of flow graphs is interesting because it deals with two fundamental phenomena in the theory of computation: memory (in the form of register or delay elements) and infinite behaviour (in the form of feedback).

© 2005 Elsevier B.V. All rights reserved.

MSC: 68Q10; 68Q55; 65A85

Keywords: Streams; Coinduction; Coalgebra; Signal flow graphs

1. Introduction

Infinite sequences or streams occur at many different places, both in mathematics and computer science, and in every day life. For the latter, think of bit streams flowing through the chips of your computer, or through the ether carrying messages from your mobile telephone. More generally, signals in the theory of signal processing are commonly represented by streams of real numbers. Also functions on streams are relevant in that setting, as they are the building blocks for filters and converters (such as the digital to analog converter in cd-players). An example of streams appearing in computer science is dataflow, which

* Tel.: +31 20 592 4116; fax: +31 20 592 4199.
E-mail address: jan.rutten@cwi.nl.
studies networks consisting of nodes and channels, through which streams of data elements flow. Streams occur also in various areas of mathematics. Examples are the use of streams in analysis: the basic notion of limit is formulated in terms of streams, and the Taylor coefficients of analytical functions (such as \((0, 1, 0, -1, 0, 1, 0, -1, \ldots)\) for \(\sin(x)\)) are streams; in combinatorics, streams are often defined by recurrence relations or difference equations, for instance representing the solution of counting problems (such as the stream of Fibonacci numbers \((1, 1, 2, 3, 5, 8, \ldots)\)); streams are used to model trajectories in dynamical systems. Many more examples exist. These notes intend to study streams as such, in principle independent of any of the afore-mentioned areas, but often taking examples from some of them, mostly from computer science. We shall develop systematic ways for: (1) defining and specifying streams (and functions on streams); (2) reasoning about streams, notably proving equalities between them; and (3) implementing streams, in particular using some basic form of (stream) circuits, known as (signal) flow graphs in the world of signal processing. All of this will involve a bit of elementary but not so standard mathematics, which will be explained in all detail along the way. In short, because streams form an infinite datatype, the mathematical techniques of algebraic specification that are traditionally used for finite data types, are not really appropriate. Instead, we shall use a relatively new proof and definition principle called coinduction, which is based on a recently developed general theory of dynamical systems called coalgebra. The contribution of the present paper lies in a new application of coinductive stream calculus to signal flow graphs. The main technical statement of this paper: the characterization of finite stream circuits in terms of so-called rational streams, is well known in the world of signal processing, where it is formulated and proved in terms of the Z-transform (a discrete version of the Laplace transform) and transfer functions. What is new here is our elementary formulation and proof of essentially the same result, by using only streams and coinduction. This elementary treatment of signal flow graphs is interesting because it explains in very basic terms two fundamental phenomena in the theory of computation: memory (in the form of register or delay elements) and infinite behaviour (in the form of feedback). Moreover, the present methodology is relevant for the area of component-based software engineering: recently, it has been generalised in order to model software composition by means of so-called component connectors in terms of relations on the streams of ingoing and outgoing messages (or data elements) at the various communication ports. Pointers to this and other related work will be discussed in the last section of this paper.

All in all, the present paper is intended as a tutorial on the basics of both a formal calculus of streams and an application thereof to signal flow graphs. The many exercises have been designed for second year undergraduate students and are rather elementary. Their main purpose is to help the not so experienced reader understand the main ideas.

2. Streams and coinduction

We introduce the set of streams (with elements in an arbitrary set \(A\)) and explain how to define streams and operations on streams by stream differential equations, and how to prove facts about streams by coinduction.
2.1. Streams

Let $A$ be any set. We define the set $A^\omega$ of all streams over $A$ as

$$A^\omega = \{\sigma \mid \sigma : \{0, 1, 2, \ldots\} \to A\}.$$  

In this section, we make no assumptions on $A$, but in some of the examples, we shall look at particular instantiations of $A$ (such as the natural numbers); in Section 3, $A$ will be the set $\mathbb{R}$ of real numbers.

For a stream $\sigma$, we call $\sigma(0)$ the initial value of $\sigma$. We define the derivative $\sigma'$ of a stream $\sigma$, for all $n \geq 0$, by

$$\sigma'(n) = \sigma(n + 1).$$

Initial value and derivative are usually called head and tail but the present terminology helps us, as we shall see, to develop a calculus of streams in close analogy to classical calculus in analysis.

Although streams will be viewed and handled as single mathematical entities, it will at various moments be convenient to refer to the individual elements of which they are made. For this, we shall use the following notation:

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

(Similarly, we shall write $\tau = (\tau(0), \tau(1), \tau(2), \ldots)$ and the like.) With this notation, the derivative of $\sigma$ is given by

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

For any $n \geq 0$, $\sigma(n)$ is called the $n$th element of $\sigma$. It can also be expressed in terms of higher-order stream derivatives, defined, for all $k \geq 0$, by

$$\sigma^{(0)} = \sigma, \quad \sigma^{(k+1)} = (\sigma^{(k)})'.

(For higher-order derivatives of order two or three, both notations will be used: we shall occasionally write $\sigma'', \sigma'''$ as well as $\sigma^{(2)}, \sigma^{(3)}$.)

**Lemma 2.1.** The $n$th element of a stream $\sigma$ is given by

$$\sigma(n) = \sigma^{(n)}(0).$$

**Exercise 2.2.** Prove Lemma 2.1 by showing by induction on $n$: for all $n \geq 0$ and for all $i \geq 0$,

$$\sigma(n + i) = \sigma^{(n)}(i).$$

We shall also use the following notation, which will be convenient when we want to compute the first few elements of a stream. For $a \in A$ and $\tau \in A^\omega$, we define

$$a : \tau = (a, \tau(0), \tau(1), \tau(2), \ldots).$$
With this notation, we have for any \( a \in A \),
\[
\sigma = \sigma(0) : \sigma' = \sigma(0) : (\sigma'(0) : \sigma^{(2)}) = \sigma(0) : (\sigma(1) : \sigma^{(2)}).
\]
Leaving out the brackets, we have, more generally for any \( n \geq 1 \),
\[
\sigma = \sigma(0) : \sigma(1) : \cdots : \sigma(n-1) : \sigma^{(n)}.
\]

2.2. Stream differential equations

A particularly convenient and succinct way of defining streams is by means of so-called stream differential equations. In analogy to differential equations from mathematical analysis, such as \( f'(x) = f(x) \), \( f(0) = 1 \), which defines the function \( f(x) = e^x \), stream differential equations specify streams (and functions on streams) in terms of their derivatives and initial values.

In case differential equations are not your favourite topic in mathematics, there is no reason to become worried at this point. No previous knowledge will be required and, more importantly, the theory of stream differential equations is much simpler and very intuitive. We shall become somewhat more formal later in this section; for now, we explain the use of stream differential equations by a number of examples.

\textbf{Example 2.3.} Let \( a \in A \) and consider the following stream differential equation:

<table>
<thead>
<tr>
<th>derivative</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma' = \sigma )</td>
<td>( \sigma(0) = a )</td>
</tr>
</tbody>
</table>

The intended interpretation of this equation is that there exists a unique stream \( \sigma \) with derivative \( \sigma' = \sigma \) and initial value \( \sigma(0) = a \). The differential equation should be read as a definition of this unique stream \( \sigma \), which is called the solution of the differential equation. (In Section 2.4, we shall return to the question whether such a unique solution always exists.)

Computing the first few elements of \( \sigma \) gives
\[
\sigma = \sigma(0) : \sigma' = a : \sigma = a : \sigma = a : a : a : \sigma.
\]
One can easily prove by induction on \( n \) that the \( n \)th derivative \( \sigma^{(n)} \) of \( \sigma \) satisfies \( \sigma^{(n)} = \sigma \). Since \( \sigma(n) = \sigma^{(n)}(0) \), by Lemma 2.1 above, it follows that
\[
\sigma(n) = \sigma^{(n)}(0) = \sigma(0) = a.
\]
In other words, \( \sigma = (a, a, a, \ldots) \).

\textbf{Example 2.4.} Let \( a, b \in A \) and consider the following differential equation:

<table>
<thead>
<tr>
<th>derivative</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma'' = \sigma )</td>
<td>( \sigma(0) = a, \sigma'(0) = b )</td>
</tr>
</tbody>
</table>

Since this equation specifies \( \sigma \) in terms of its second derivative \( \sigma'' (= \sigma^{(2)}) \), we call it a \textit{higher-order} differential equation. Note that it not only specifies the initial value of \( \sigma \) but
also that of \( \sigma' \). Computing again the first few values, we find

\[
\sigma = \sigma(0) : \sigma' = a : \sigma'(0) : \sigma'' = a : b : \sigma = a : b : a : b : \sigma.
\]

Thus \( \sigma = (a, b, a, b, a, b, \ldots) \). Instead of a higher-order differential equation, we can also use a system of differential equations to define this stream \( \sigma \):

<table>
<thead>
<tr>
<th>derivative</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma' = \tau )</td>
<td>( \sigma(0) = a )</td>
</tr>
<tr>
<td>( \tau' = \sigma )</td>
<td>( \tau(0) = b )</td>
</tr>
</tbody>
</table>

This system of equations defines two streams \( \sigma \) and \( \tau \), with \( \sigma \) as before and with \( \tau = (b, a, b, a, b, a, \ldots) \).

**Exercise 2.5.** Let \( a, b, c \in A \). Define the following streams by means of a differential equation (or a system of differential equations):

(i) \( \sigma = (a, b, a, b, a, b, \ldots) \),

(ii) \( \sigma = (a, a, b, a, a, b, a, b, \ldots) \),

(iii) \( \sigma = (a, b, b, c, c, c, a, b, b, c, c, c, \ldots) \),

(iv) Try the same for an arbitrary stream \( \sigma \).

So far we have used stream differential equations to define individual streams, such as \( (a, b, a, b, a, b, \ldots) \). We use differential equations also for the definition of *functions* on streams. Here are again some examples.

**Example 2.6.** Consider the following differential equation:

<table>
<thead>
<tr>
<th>derivative</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{even}(\sigma)' = \text{even}(\sigma'') )</td>
<td>( \text{even}(\sigma)(0) = \sigma(0) )</td>
</tr>
</tbody>
</table>

The intended meaning of this equation is that there exists for every stream \( \sigma \) a unique stream called \( \text{even}(\sigma) \) such that \( \text{even}(\sigma)' = \text{even}(\sigma'') \) and \( \text{even}(\sigma)(0) = \sigma(0) \). This single equation is in fact an infinite system of equations, one for each \( \sigma \in A^\omega \). All these infinitely many equations together define a function

\[ \text{even} : A^\omega \to A^\omega \]

that assigns to a stream \( \sigma \) the unique stream \( \text{even}(\sigma) \) specified by these equations. How does \( \text{even}(\sigma) \), for a given stream \( \sigma \), look like? Computing the first few values gives

\[
\begin{align*}
\text{even}(\sigma) &= \text{even}(\sigma)(0) : \text{even}(\sigma)' \\
&= \sigma(0) : \text{even}(\sigma'') \\
&= \sigma(0) : \text{even}(\sigma''')(0) : \text{even}(\sigma''')' \\
&= \sigma(0) : \sigma(2) : \text{even}(\sigma^{(4)})
\end{align*}
\]

and so on. By induction on \( n \), we can prove, for any \( n \geq 0 \), that

\[
(\text{even}(\sigma))^{(n)} = \text{even}(\sigma^{(2n)}).
\]

(1)
It follows that

\[
\begin{align*}
\text{even}(\sigma)(n) &= \text{even}(\sigma(2n))(0) \\
&= \text{even}(\sigma(2n))(0) \\
&= \sigma(2n)(0) \\
&= \sigma(2n).
\end{align*}
\]

Thus the function even maps a stream \(\sigma\) to

\[
\text{even}(\sigma) = (\sigma(0), \sigma(2), \sigma(4), \ldots)
\]
as its name already suggested it would.

**Exercise 2.7.** Prove identity (1) above.

**Exercise 2.8.** Consider the following differential equation:

\[
\begin{align*}
\text{derivative} \quad &\text{initial value} \\
\text{odd}(\sigma)' = \text{odd}(\sigma'') &\text{odd}(\sigma)(0) = \sigma(1)
\end{align*}
\]

Show that it defines a function odd : \(A^\omega \rightarrow A^\omega\) that maps a stream \(\sigma\) to the stream

\[
\text{odd}(\sigma) = (\sigma(1), \sigma(3), \sigma(5), \ldots).
\]

How are the functions even and odd related?

**Example 2.9.** Let the function zip : \(A^\omega \times A^\omega \rightarrow A^\omega\) be defined by the following differential equation (more precisely: system of differential equations, one for every \(\sigma\) and \(\tau\) in \(A^\omega\)):

\[
\begin{align*}
\text{derivative} \quad &\text{initial value} \\
\text{zip}(\sigma, \tau)' = \text{zip}(\tau, \sigma') &\text{zip}(\sigma, \tau)(0) = \sigma(0)
\end{align*}
\]

For given \(\sigma, \tau \in A^\omega\), the first few values of the stream \(\text{zip}(\sigma, \tau)\) are:

\[
\begin{align*}
\text{zip}(\sigma, \tau) &= \text{zip}(\sigma, \tau)(0) : \text{zip}(\sigma, \tau)' \\
&= \sigma(0) : \text{zip}(\tau, \sigma') \\
&= \sigma(0) : \text{zip}(\tau, \sigma')(0) : \text{zip}(\tau, \sigma')' \\
&= \sigma(0) : \tau(0) : \text{zip}(\sigma', \tau') \\
&= \sigma(0) : \tau(0) : \text{zip}(\sigma', \tau')(0) : \text{zip}(\sigma', \tau')' \\
&= \sigma(0) : \tau(0) : \sigma(1) : \text{zip}(\tau', \sigma'').
\end{align*}
\]

Continuing this way, we see (and can prove formally by induction):

\[
\text{zip}(\sigma, \tau) = (\sigma(0), \tau(0), \sigma(1), \tau(1), \sigma(2), \tau(2), \ldots).
\]

**Exercise 2.10.** (a) Define by means of a higher-order differential equation a function double : \(A^\omega \rightarrow A^\omega\) that maps a stream \(\sigma\) to

\[
\text{double}(\sigma) = (\sigma(0), \sigma(0), \sigma(1), \sigma(1), \sigma(2), \sigma(2), \ldots).
\]
(b) How are the functions `double` and `zip` related?

2.3. Proofs by coinduction

Recall the streams `(a, a, a, . . .)` and `(a, b, a, b, a, b, . . .)` defined in Examples 2.3 and 2.4, and let us call them `[a]` and `[ab]`, respectively. They satisfy

```
[a] = a : [a],    [ab] = a : b : [ab].
```

Now suppose that we want to prove the following equality:

```
even([ab]) = [a].
```

One could argue that this identity is so trivial that no proof is needed. But for one thing, we want to be really precise and formal, and furthermore, we shall see many other examples that are much more complicated. So recalling the definition of the function `even : A^o \rightarrow A^o` from Example 2.6, we compute the first few values of `even([ab])`:

```
even([ab]) = even([ab])(0) : even([ab])' = [ab](0) : even([ab])'' = a : even([ab]).
```

Comparing this with `[a] = a : [a]`, we see that the initial values of both streams are equal: `even([ab])(0) = a = [a](0)`. If we can prove that their derivatives are equal too, then we can conclude that `even([ab]) = [a]`. Since `even([ab])' = even([ab])` and `[a]' = [a]`, we see that in order to prove `even([ab]) = [a]`, we have to prove `even([ab]) = [a]`. Thus our reasoning seems to be trapped in a vicious circle.

Still, the two equalities `even([ab]) = a : even([ab])` and `[a] = a : [a]` taken together seem to leave no doubt about the validity of `even([ab]) = [a]`, since they allow us to prove that both streams agree on initial segments of arbitrary length.

Below we introduce a proof method, called coinduction, that does allow us to give a formal proof of identities such as the one above. It will be formulated in terms of the following notion:

**Definition 2.11.** A bisimulation on `A^o` is a relation `R \subseteq A^o \times A^o` such that, for all `\sigma` and `\tau` in `A^o`,

```
if \langle \sigma, \tau \rangle \in R then \begin{cases} 
  (i) & \sigma(0) = \tau(0) \\
  (ii) & \langle \sigma', \tau' \rangle \in R.
\end{cases}
```

(We shall sometimes write `\sigma R \tau` for `\langle \sigma, \tau \rangle \in R`.)

If there exists a bisimulation relation `R` with `\sigma R \tau` then we write `\sigma \sim \tau` and say that `\sigma` and `\tau` are bisimilar. In other words, the bisimilarity relation `\sim` is the union of all bisimulations:

```
\sim = \bigcup \{ R \subseteq A^o \times A^o \mid R \text{ is a bisimulation relation} \}.
```
Lemma 2.12. If $R \subseteq A^\omega \times A^\omega$ and $S \subseteq A^\omega \times A^\omega$ are bisimulations then both the union $R \cup S$ and the relational composition

$$R \circ S = \{(\sigma, \tau) \in A^\omega \times A^\omega \mid \exists \rho \in A^\omega, (\sigma, \rho) \in R \land (\rho, \tau) \in S\}$$

of $R$ and $S$ are again bisimulation relations.

Exercise 2.13. Prove Lemma 2.12. Also prove that the bisimilarity relation $\sim$ is itself a bisimulation relation.

Theorem 2.14 (Coinduction). For all $\sigma, \tau \in A^\omega$, if there exists a bisimulation relation $R \subseteq A^\omega \times A^\omega$ with $(\sigma, \tau) \in R$, then $\sigma = \tau$. In other words

$$\sigma \sim \tau \Rightarrow \sigma = \tau.$$

Proof. Consider two streams $\sigma$ and $\tau$ and let $R \subseteq A^\omega \times A^\omega$ be a bisimulation on $A^\omega$ containing the pair $(\sigma, \tau)$. It follows by induction on $n$ that $(\sigma(n), \tau(n)) \in R$, for all $n \geq 0$, because $R$ is a bisimulation. This implies, again because $R$ is a bisimulation, that $\sigma(n)(0) = \tau(n)(0)$, for all $n \geq 0$. By Lemma 2.1, $\sigma(n) = \tau(n)$, for all $n \geq 0$. Now $\sigma = \tau$ follows. \(\square\)

Exercise 2.15. Show that the converse of Theorem 2.14 also holds: for all $\sigma, \tau \in A^\omega$, if $\sigma = \tau$ then $\sigma \sim \tau$. (Hint: show that $\{(\sigma, \sigma) \mid \sigma \in A^\omega\}$ is a bisimulation relation on $A^\omega$.)

In order to prove the equality of two streams $\sigma$ and $\tau$, it is according to Theorem 2.14 sufficient to establish the existence of a bisimulation relation $R \subseteq A^\omega \times A^\omega$ with $(\sigma, \tau) \in R$. This proof principle is called coinduction (and is sometimes also referred to as the bisimulation proof method). The coinduction proof principle can be seen as a systematic way of strengthening the statement one is trying to prove: instead of proving only the single identity $\sigma = \tau$, one computes the smallest bisimulation relation $R$ that contains the pair $(\sigma, \tau)$. The construction of $R$ is always the same, and amounts to the computation of the closure of $\{(\sigma, \tau)\}$ under taking derivatives; at every stage of the construction of $R$, one should check that the initial values of the streams in newly added pairs are equal. By the coinduction proof principle, it follows that $\alpha = \beta$ for all pairs $(\alpha, \beta) \in R$. Since $(\sigma, \tau) \in R$, by the construction of $R$, it follows in particular that $\sigma = \tau$.

Example 2.16. We prove the equality $\text{even}([ab]) = [a]$, which we discussed at the beginning of this section, by coinduction. We define $R = \{([\text{even}([ab])], [a])\}$. In order to show that $R$ is a bisimulation, we have to check for the pair $(\text{even}([ab]), [a])$ that it satisfies the two bisimulation conditions of Definition 2.11:

(i) $\text{even}([ab])(0) = [a](0)$ and (ii) $([\text{even}([ab])]', [a]') \in R$.

These follow from $\text{even}([ab])(0) = a = [a](0)$, $\text{even}([ab])' = \text{even}([ab])$, and $[a]' = [a]$. Now $\text{even}([ab]) = [a]$ follows by coinduction (Theorem 2.14).

Example 2.17. We prove by coinduction: for all $\sigma, \tau \in A^\omega$,

$$\text{even}(\text{zip}(\sigma, \tau)) = \sigma.$$
equality

The relation coinduction. This is the right way to start proofs by coinduction in general. Often, however, $R$
As it turned out, $R$

(The function $\text{zip}$ was defined in Example 2.9.) In order to find a suitable bisimulation, we compute, for given $\sigma$ and $\tau$,

\[
even(\text{zip}(\sigma, \tau)) = even(\text{zip}(\sigma, \tau))(0) : even(\text{zip}(\sigma, \tau))' = \text{zip}(\sigma, \tau)(0) : even(\text{zip}(\sigma, \tau))'' = \sigma(0) : even(\text{zip}(\sigma', \tau')).
\]

Comparing this to $\sigma = \sigma(0) : \sigma'$ suggests the following definition for a relation $R \subseteq A^\omega \times A^\omega$:

\[
R = \{ (\text{even}(\text{zip}(\sigma, \tau)), \sigma) \mid \sigma, \tau \in A^\omega \}.
\]

Now $even(\text{zip}(\sigma, \tau))(0) = \sigma(0)$ and $even(\text{zip}(\sigma, \tau))' = even(\text{zip}(\sigma', \tau'))$, for any $\sigma, \tau \in A^\omega$, implies $\langle even(\text{zip}(\sigma, \tau))', \sigma' \rangle \in R$. As a consequence, $R$ is a bisimulation. The equality $even(\text{zip}(\sigma, \tau)) = \sigma$ now follows by coinduction.

That was easy. We defined $R$ simply as the set of all pairs that we wanted to prove equal. As it turned out, $R$ was a bisimulation and the identity we were looking after followed by coinduction. This is the right way to start proofs by coinduction in general. Often, however, the relation $R$ has to be extended further before it satisfies condition (ii) of the definition of bisimulation. In other words, the relation has to be closed under taking derivatives. Everytime that new pairs are added to the relation, conditions (i) and (ii) of the definition of bisimulation have to be checked again. The following example clearly illustrates what we mean.

**Example 2.18.** We prove by coinduction: for all $\sigma \in A^\omega$,

\[
\text{zip}(\text{even}(\sigma), \text{odd}(\sigma)) = \sigma.
\]  

(The function $\text{odd}$ was introduced in Exercise 2.8.) As before, we begin by defining

\[
R = \{ (\text{zip}(\text{even}(\sigma), \text{odd}(\sigma)), \sigma) \mid \sigma \in A^\omega \}.
\]

In order to check that $R$ satisfies the bisimulation conditions (i) and (ii) of Definition 2.11, we compute:

\[
\text{zip}(\text{even}(\sigma), \text{odd}(\sigma)) = \text{zip}(\text{even}(\sigma), \text{odd}(\sigma))(0) : \text{zip}(\text{even}(\sigma), \text{odd}(\sigma))' = \sigma(0) : \text{zip}(\text{odd}(\sigma), \text{even}(\sigma')).
\]

For all pairs $\langle \text{zip}(\text{even}(\sigma), \text{odd}(\sigma)), \sigma \rangle \in R$, we see that the initial values agree:

\[
\text{zip}(\text{even}(\sigma), \text{odd}(\sigma))(0) = \sigma(0)
\]

but that the pair of derivatives:

\[
\langle \text{zip}(\text{even}(\sigma), \text{odd}(\sigma))', \sigma' \rangle = \langle \text{zip}(\text{odd}(\sigma), \text{even}(\sigma'))', \sigma' \rangle
\]
is not in $R$. Therefore, we extend the relation $R$ by including these latter pairs as well. Thus our second proposal for $R$ now looks like

$$R = \{ \langle \text{zip}(\text{even}(\sigma), \text{odd}(\sigma)), \sigma \rangle \mid \sigma \in A^\omega \}$$

$$\cup \{ \langle \text{zip}(\text{odd}(\sigma), \text{even}(\sigma')), \sigma' \rangle \mid \sigma \in A^\omega \}.$$ 

Next it should be checked whether the two bisimulation conditions (i) and (ii) are satisfied by all the newly added pairs $\langle \text{zip}(\text{odd}(\sigma), \text{even}(\sigma')), \sigma' \rangle$. We compute:

$$\text{zip}(\text{odd}(\sigma), \text{even}(\sigma'))(0) = \sigma(1) = \sigma'(0).$$

The pair of derivatives now is in $R$:

$$\langle \text{zip}(\text{odd}(\sigma), \text{even}(\sigma'))', \sigma'' \rangle = \langle \text{zip}(\text{even}(\sigma'), \text{odd}(\sigma'')), \sigma'' \rangle \in R.$$ 

This concludes the construction of $R$ and the proof that it is a bisimulation. Identity (2) now follows by coinduction.

**Exercise 2.19.** Prove by coinduction that $\text{odd}(\text{zip}(\sigma, \tau)) = \tau$, for all $\sigma, \tau \in A^\omega$.

### 2.4. Solutions of differential equations

We have seen many examples of how stream differential equations can be used for the definition of streams and stream functions. Here we give a sketch of a proof that such (systems of) stream differential equations have a unique solution. The proof will be based on the fact that the set $A^\omega$ of all streams can be turned into a so-called final stream automaton, a notion which is introduced first.

A stream automaton (also called stream coalgebra) is a triple

$$\langle Q, o_Q : Q \to A, t_Q : Q \to Q \rangle$$

(sometimes also denoted by $\langle Q, o_Q, t_Q \rangle$) consisting of a set $Q$ of states, an output function $o_Q : Q \to A$, and a transition function $t_Q : Q \to Q$. For two stream automata $\langle P, o_P, t_P \rangle$ and $\langle Q, o_Q, t_Q \rangle$, a function $f : P \to Q$ is a homomorphism, denoted by

$$f : \langle P, o_P, t_P \rangle \to \langle Q, o_Q, t_Q \rangle$$

whenever, for all $p \in P$, $o_P(p) = o_Q(f(p))$ and $f(t_P(p)) = t_Q(f(p))$. The set $A^\omega$ of all streams can itself be turned into a stream automaton as follows. Defining $o : A^\omega \to A$ by $o(\sigma) = \sigma(0)$ and $t : A^\omega \to A^\omega$ by $t(\sigma) = \sigma'$, we obtain a stream automaton $\langle A^\omega, o, t \rangle$. It has the following universal property.
Theorem 2.20. The automaton \( \langle A^\omega, o, t \rangle \) is final among the family of all stream automata. That is, for any automaton \( \langle Q, o_Q, t_Q \rangle \) there exists a unique homomorphism \( l : \langle Q, o_Q, t_Q \rangle \rightarrow \langle A^\omega, o, t \rangle \).

Proof. Let \( \langle Q, o_Q, t_Q \rangle \) be a stream automaton and define \( l : Q \rightarrow A^\omega \) as
\[
l(q) = (o(q), o(t(q)), o(t(t(q))), \ldots)
\]
for \( q \) in \( Q \). It is straightforward to show that \( l \) is a homomorphism from \( \langle Q, o_Q, t_Q \rangle \) to \( \langle A^\omega, o, t \rangle \). For uniqueness, suppose \( f \) and \( g \) are homomorphisms from \( Q \) to \( A^\omega \). The equality of \( f \) and \( g \) follows by coinduction from the fact that \( R = \{(f(q), g(q)) \mid q \in Q\} \) is a bisimulation on \( A^\omega \), which is proved next. Consider \( (f(q), g(q)) \in R \). Because \( f \) and \( g \) are homomorphisms, \( o(f(q)) = o(g(q)) \). Furthermore, \( t(f(q)) = f(t_Q(q)) \) and \( t(g(q)) = g(t_Q(q)) \). Because \( f(t_Q(q)) \) and \( g(t_Q(q)) \) are homomorphisms, \( f(t_Q(q)) = g(t_Q(q)) \). By Theorem 2.20, there exists a unique homomorphism \( \mathsf{zip} \) from \( Q, o_Q, t_Q \) to \( A^\omega \) as \( R = \{(o(q), o(t(q))) \mid q \in Q\} \) is a bisimulation on \( A^\omega \), and this shows that \( R \) is a bisimulation. Now \( f = g \) follows by the coinduction proof principle Theorem 2.14.

By the finiteness of the automaton \( \langle A^\omega, o, t \rangle \) we can prove for many stream differential equations that they have a unique solution (thereby justifying their use as definitions). We present a few examples (see [12] for general results).

Example 2.21. Recall our first stream differential equation:

<table>
<thead>
<tr>
<th>derivative</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma' = \sigma )</td>
<td>( \sigma(0) = a )</td>
</tr>
</tbody>
</table>

Consider a stream automaton \( \langle Q, o_Q, t_Q \rangle \) consisting of a singleton set \( Q = \{q\} \) with \( o_Q(q) = a \) and \( t_Q(q) = q \). By Theorem 2.20, there exists a unique homomorphism \( l : Q \rightarrow A^\omega \). We now define \( \sigma = l(q) \). Because \( l \) is a homomorphism, it follows that \( \sigma \) is a solution of the differential equation: \( \sigma' = t(\sigma) = t(l(q)) = l(t_Q(q)) = l(q) = \sigma \) and \( \sigma(0) = o(\sigma) = o(l(q)) = o_Q(q) = a \). If \( \tau \) is a stream with \( \tau' = \tau \) and \( \tau(0) = a \), then \( \sigma = \tau \) follows, by the coinduction proof principle Theorem 2.14, from the fact that \( \{\langle \sigma, \tau \rangle \} \) is a bisimulation relation on \( A^\omega \). Thus \( \sigma \) is the only solution of the differential equation.

Example 2.22. In order to prove that the system of stream differential equations of Example 2.9 has a unique solution, and therefore uniquely determines the function \( \mathsf{zip} : A^\omega \times A^\omega \rightarrow A^\omega \), we consider a stream automaton \( \langle Q, o_Q, t_Q \rangle \), with \( Q = A^\omega \times A^\omega \) and with, for all \( \langle \sigma, \tau \rangle \in Q \),
\[
t_Q(\langle \sigma, \tau \rangle) = (\tau, \sigma'), \quad o_Q(\langle \sigma, \tau \rangle) = \sigma(0).
\]
As before, there exists by Theorem 2.20, a unique homomorphism \( l : Q \rightarrow A^\omega \). We now define \( \mathsf{zip}(\sigma, \tau) = l(\langle \sigma, \tau \rangle) \). Similar to the first example, it is not difficult to prove that \( \mathsf{zip} \) is the unique function satisfying the above system of stream differential equations.

The first example above involved a stream automaton with only one state; in the second example, an infinite automaton was used. One can, more generally, show the existence of a unique solution for more complicated systems of stream differential equations by using more
complicated stream automata. All the stream differential equations that we shall encounter in these notes, can be shown to have a unique solution in this manner. We shall not present any details here and refer the interested reader to [12].

3. Basic stream calculus

In this section, we study the set $\mathbb{R}^{*}$ of streams of real numbers. We shall introduce a number of constants and shall define the operations of sum, product, and inverse of streams. These constants and operations make of $\mathbb{R}^{*}$ a calculus with many pleasant properties. For instance, it will be possible to compute solutions of linear systems of equations.

3.1. Sum and product

Let $\sigma$ and $\tau$ be two streams of real numbers. For notational convenience, we shall sometimes denote the elements of $\sigma$ and $\tau$ by $\sigma_n$ and $\tau_n$ instead of $\sigma(n)$ and $\tau(n)$:

$$\sigma = (\sigma_0, \sigma_1, \sigma_2, \ldots), \quad \tau = (\tau_0, \tau_1, \tau_2, \ldots).$$

**Definition 3.1.** We define the sum $\sigma + \tau$ of $\sigma$ and $\tau$ by

$$\sigma + \tau = (\sigma_0 + \tau_0, \sigma_1 + \tau_1, \sigma_2 + \tau_2, \ldots).$$

(Notice that we use the same symbol $+$ for both the sum of two streams and the sum of two real numbers.) We define the convolution product $\sigma \times \tau$ of $\sigma$ and $\tau$ by

$$\sigma \times \tau = (\sigma_0 \cdot \tau_0, (\sigma_0 \cdot \tau_1) + (\sigma_1 \cdot \tau_0), (\sigma_0 \cdot \tau_2) + (\sigma_1 \cdot \tau_1) + (\sigma_2 \cdot \tau_0), \ldots).$$

That is, for any $n \geq 0$,

$$(\sigma \times \tau)(n) = (\sigma_0 \cdot \tau_n) + (\sigma_1 \cdot \tau_{n-1}) + \cdots + (\sigma_{n-1} \cdot \tau_1) + (\sigma_n \cdot \tau_0)
= \sum_{k=0}^{n} \sigma_k \cdot \tau_{n-k}.$$  

In general, we shall simply say ‘product’ rather than ‘convolution product’. Note that we use the symbol $\times$ for the multiplication of streams and the symbol $\cdot$ for the multiplication of real numbers.

Similar to the notation for the multiplication of real numbers (and functions), we shall write

$$\sigma^0 \equiv 1, \quad \sigma^{n+1} \equiv \sigma \times \sigma^n$$

for any $\sigma \in \mathbb{R}^{*}$ and $n \geq 0$. Note the distinction between this notation and the notation $\sigma^{(n)}$ for the $n$th derivative of $\sigma$ that was introduced in Section 2.
The sum of $\sigma$ and $\tau$ consists of the stream of the sum of their elements. In order to understand the product $\sigma \times \tau$, think of a stream $\sigma$ as a process producing its respective elements $\sigma_0, \sigma_1, \sigma_2$, and so on, one by one in an infinite sequence of time steps. The product $\sigma \times \tau$ can then be viewed as a kind of delayed computation of the elementwise product, in the following sense:

$$\sigma \times \tau = (\sigma_0 \cdot \tau_0, \sigma_0 \cdot \tau_1, \sigma_0 \cdot \tau_2, \sigma_0 \cdot \tau_3, \ldots)$$
$$+ (0, \sigma_1 \cdot \tau_0, \sigma_1 \cdot \tau_1, \sigma_1 \cdot \tau_2, \ldots)$$
$$+ (0, 0, \sigma_2 \cdot \tau_0, \sigma_2 \cdot \tau_1, \ldots)$$
$$+ \ldots$$

It will be convenient to define the operations of sum and product also for the combination of a real number $r$ and a stream $\sigma$. This will allow us, for instance, to write $3 \times \sigma$ for $\sigma + \sigma + \sigma$. In order to define this formally, it will be convenient to view real numbers as streams in the following manner:

**Definition 3.2.** We define for every $r \in \mathbb{R}$ a stream $[r] \in \mathbb{R}^{\omega}$ by

$$[r] = (r, 0, 0, 0, \ldots).$$

Note that this defines in fact a function

$$[\cdot] : \mathbb{R} \rightarrow \mathbb{R}^{\omega}, \quad r \mapsto [r]$$

which embeds the set of real numbers into the set of streams.

This definition allows us to add and multiply real numbers $r$ with streams $\sigma$, yielding:

$$[r] + \sigma = (r, 0, 0, 0, \ldots) + \sigma$$
$$= (r + \sigma_0, \sigma_1, \sigma_2, \sigma_3 \ldots),$$

$$[r] \times \sigma = (r, 0, 0, 0, \ldots) \times \sigma$$
$$= (r \cdot \sigma_0, r \cdot \sigma_1, r \cdot \sigma_2, \ldots).$$

**Notation 3.3.** For notational convenience, we shall very quickly start to simply write $r + \sigma$ for $[r] + \sigma$, and similarly $r \times \sigma$ for $[r] \times \sigma$. The context will always make clear whether the notation $r$ has to be interpreted as the real number $r$ or as the stream $[r]$. For multiplication, this difference is moreover made explicit by the use of two different symbols: $r \times \sigma$ always denotes the multiplication of streams (and hence $r$ should be read as the stream $[r]$) and $r \cdot s$ always denotes the multiplication of real numbers. We shall also use the following convention:

$$-\sigma = [-1] \times \sigma$$
$$= (-\sigma_0, -\sigma_1, -\sigma_2, \ldots).$$

Next we present a few basic properties of our operators.
**Proposition 3.4.** For all $r, s \in \mathbb{R}$ and $\sigma, \tau, \rho \in \mathbb{R}^\omega$,

\[
[r] + [s] = [r + s],
\]
\[
\sigma + 0 = \sigma,
\]
\[
\sigma + \tau = \tau + \sigma,
\]
\[
\sigma + (\tau + \rho) = (\sigma + \tau) + \rho,
\]
\[
[r] \times [s] = [r \cdot s],
\]
\[
0 \times \sigma = 0,
\]
\[
1 \times \sigma = \sigma,
\]
\[
\sigma \times \tau = \tau \times \sigma,
\]
\[
\sigma \times (\tau + \rho) = (\sigma \times \tau) + (\sigma \times \rho),
\]
\[
\sigma \times (\tau \times \rho) = (\sigma \times \tau) \times \rho.
\]

**Exercise 3.5.** Prove the equalities given in Proposition 3.4.

3.2. Polynomial streams

Particularly simple are those streams that from a certain point onwards are constant zero:

\[
\sigma = (r_0, r_1, r_2, \ldots, r_n, 0, 0, 0, \ldots)
\]

for $n \geq 0$ and $r_0, \ldots, r_n \in \mathbb{R}$. Using the following constant, we shall see that there is a very convenient way of denoting such streams.

**Definition 3.6.** We define the constant $X$ by

\[
X = (0, 1, 0, 0, 0, \ldots).
\]

**Proposition 3.7.** For all $r \in \mathbb{R}$, $\sigma \in \mathbb{R}^\omega$, and $n \geq 0$:

\[
r \times X = (0, r, 0, 0, 0, \ldots),
\]
\[
X \times \sigma = (0, \sigma_0, \sigma_1, \sigma_2, \ldots),
\]
\[
X^n = \underbrace{(0, \ldots, 0, 1, 0, 0, 0, \ldots)}_{n \text{ times}}.
\]

**Exercise 3.8.** Prove Proposition 3.7.

**Exercise 3.9.** (a) Compute $1 - 2X + 5X^4 + X^5$ and $1 - X + X^2 - X^3 + X^4$.

(b) Write $(1, 1, 1, 1, 0, 0, 0, \ldots)$ using sum, product, and $X$.

(c) Same question for $(1, 0, -1, 0, 1, 0, -1, 0, 0, 0, \ldots)$.

(d) Can you do the same for $(1, 1, 1, \ldots)$, the infinite stream of ones?

**Definition 3.10.** For all $n \geq 0$ and all $r_0, \ldots, r_n \in \mathbb{R}$:

\[
r_0 + r_1 X + r_2 X^2 + \cdots + r_n X^n = (r_0, r_1, r_2, \ldots, r_n, 0, 0, 0, \ldots).
\]

Such streams are called **polynomial streams**.
(The equality in Definition 3.10 is an immediate consequence of the definitions of sum, product, and X.) Let us stress the fact that although a polynomial stream such as

\[ 2 + 3X - 8X^3 \]

looks like a (polynomial) function \( f(x) = 2 + 3x - 8x^3 \), for which \( x \) is a variable, it really is a stream, built from constant streams \( (2, 3, 8, \text{and } X) \), and the operations of sum and product. At the same time, it is true that we can calculate with polynomial streams in precisely the same way as we are used to compute with (polynomial) functions, as is illustrated by the following examples.

**Exercise 3.11.** Compute the following streams:

(i) \((1 + X) \times (1 - X)\),
(ii) \((1 + 7X^2) \times (-X + X^3 + 3X^4)\),
(iii) \((1 + X)^2, (1 + X)^3\), etc.

### 3.3. Derivatives of sum and product

We show how to compute the derivatives of the sum and the product of two streams, and present some examples of stream differential equations (introduced in Section 2.2) that now involve the use of sum and product.

**Theorem 3.12.** For all \( \sigma, \tau \in \mathbb{R}^\omega \),

\[
\begin{align*}
(\sigma + \tau)(0) &= \sigma_0 + \tau_0, \\
(\sigma + \tau)' &= \sigma' + \tau', \\
(\sigma \times \tau)(0) &= \sigma_0 \cdot \tau_0, \\
(\sigma \times \tau)' &= (\sigma_0 \times \tau') + (\sigma' \times \tau).
\end{align*}
\]

**Proof.** The first three equalities are immediate from Definition 3.1. For the last equality, we have

\[
\begin{align*}
(\sigma \times \tau)' &= ((\sigma_0 \cdot \tau_1) + (\sigma_1 \cdot \tau_0), (\sigma_0 \cdot \tau_2) + (\sigma_1 \cdot \tau_1) + (\sigma_2 \cdot \tau_0), \ldots) \\
&= (\sigma_0 \cdot \tau_1, \sigma_0 \cdot \tau_2, \sigma_0 \cdot \tau_3, \ldots) + (\sigma_1 \cdot \tau_0, \sigma_1 \cdot \tau_1 + (\sigma_2 \cdot \tau_0), \ldots) \\
&= \sigma_0 \times (\tau_1, \tau_2, \tau_3, \ldots) + ((\sigma_1, \sigma_2, \sigma_3, \ldots) \times (\tau_0, \tau_1, \tau_2, \ldots)) \\
&= (\sigma_0 \times \tau') + (\sigma' \times \tau). \quad \square
\end{align*}
\]

**Remark 3.13.** Sum and product of two streams \( \sigma \) and \( \tau \) satisfy, in other words, the following stream differential equations:

<table>
<thead>
<tr>
<th>derivative</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\sigma + \tau)' = \sigma' + \tau')</td>
<td>((\sigma + \tau)(0) = \sigma_0 + \tau_0)</td>
</tr>
<tr>
<td>((\sigma \times \tau)' = (\sigma_0 \times \tau') + (\sigma' \times \tau))</td>
<td>((\sigma \times \tau)(0) = \sigma_0 \cdot \tau_0)</td>
</tr>
</tbody>
</table>
These equalities can be seen as properties of the operations of sum and product, which were defined in Definition 3.1. Alternatively, they could also be taken as the definition of the operations of sum and product. From this definition, one could actually prove the equalities of Definition 3.1 as properties (which is omitted here). In short, Theorem 3.12 and Definition 3.1 can be seen as two different, but equivalent definitions.

The following equalities will be particularly helpful in some of the calculation that will follow later.

**Corollary 3.14.** For all \( r \in \mathbb{R} \) and \( \sigma \in \mathbb{R}^o \),

\[
[r](0) = r, \\
r' = [0], \\
X(0) = 0, \\
X' = [1], \\
([r] + \sigma)(0) = r + \sigma_0, \\
([r] + \sigma)' = \sigma', \\
([r] \times \sigma)(0) = r \cdot \sigma_0, \\
([r] \times \sigma)' = [r] \times \sigma', \\
(X \times \sigma)(0) = 0, \\
(X \times \sigma)' = \sigma.
\]

**Proof.** In order to avoid any confusion, we have, temporarily, denoted the stream interpretation of a real number \( r \) explicitly again by \([r]\). All of the above identities can be proved straightforwardly. For instance,

\[
X' = (0, 1, 0, 0, 0, \ldots)', \\
= (1, 0, 0, 0, \ldots), \\
= [1]
\]

to mention one example. \( \square \)

**Example 3.15.**

\[
(X \times X)' = X, \\
(X^{n+1})' = X^n, \\
(2 + 3X - 7X^3)' = 3 - 7X^2, \\
(1 - X + X^2 - X^3 + X^4)' = -1 + X - X^2 + X^3.
\]

It is clear from the above that taking the stream derivative of the product of two streams follows rules that are different from what we are used to in analysis. For (differentiable real-valued) functions \( f(x) \) and \( g(x) \), one has \((f \times g)' = (f' \times g) + (f \times g')\). (Here \( f \times g \) is defined, for all \( x \), by \((f \times g)(x) = f(x) \cdot g(x)\).) In particular, if \( f(x) = r_0 + r_1x + r_2x^2 + \cdots + r_{n+1}x^{n+1} \), then \( f'(x) = r_1 + 2r_2x + \cdots + (n + 1)r_{n+1}x^n \). We see that the rules for the computation of stream derivatives are, in fact, simpler.
**Exercise 3.16.** Compute the derivatives of the following streams:

(i) \( \sigma = 7 \),
(ii) \( \sigma = -5 - X^2 - X^6 + X^{100} \),
(iii) \( \sigma = 1 + X, \sigma = (1 + X)^2, \sigma = (1 + X)^3 \).

Can you find a formula for the derivative of \( \sigma = (1 + X)^n \), for arbitrary \( n \geq 0 \)? (Use the formula for \( (\sigma \times \tau)' \) from Theorem 3.12.)

**Exercise 3.17.** Some of the proofs of the properties listed in Proposition 3.4 are a bit awkward. Using Theorem 3.12, we can alternatively prove some of these properties more conveniently by coinduction.

(a) *Warming up:* Prove \( \sigma + \tau = \tau + \sigma \), for all \( \sigma, \tau \in \mathbb{R}^\omega \), by coinduction.

(b) *More difficult:* For \( \sigma, \tau, \rho \in \mathbb{R}^\omega \), compute and compare the derivatives of \( \sigma \times (\tau \times \rho) \) and \( (\sigma \times \tau) \times \rho \). Define a bisimulation relation which allows you to prove the associativity of \( \times \) by coinduction.

Using the properties above, we next look at some examples of streams defined by stream differential equations.

**Example 3.18.** Let \( \sigma \in \mathbb{R}^\omega \) be defined by the following stream differential equation:

<table>
<thead>
<tr>
<th>derivative</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma' = 2 \times \sigma )</td>
<td>( \sigma(0) = 1 )</td>
</tr>
</tbody>
</table>

We compute the first few elements of \( \sigma \):

\[
\begin{align*}
\sigma &= \sigma(0) : \sigma' \\
&= 1 : (2 \times \sigma) \\
&= 1 : (2 \times \sigma)(0) : (2 \times \sigma)' \\
&= 1 : 2 : (2 \times \sigma') \quad \text{(using Corollary 3.14).} \\
&= 1 : 2 : (2^2 \times \sigma) \\
&= 1 : 2 : 2^2 : (2^3 \times \sigma)
\end{align*}
\]

and so on. We see: \( \sigma = (1, 2, 2^2, 2^3, 2^4, \ldots) \).

**Exercise 3.19.** Show that \( \sigma = (1, 2, 2^2, 2^3, 2^4, \ldots) \) satisfies the differential equation of Example 3.18.

**Example 3.20.** Let \( \sigma, \tau \in \mathbb{R}^\omega \) be defined by the following equations:

<table>
<thead>
<tr>
<th>derivative</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma' = \sigma + \tau )</td>
<td>( \sigma(0) = 0 )</td>
</tr>
<tr>
<td>( \tau' = \tau )</td>
<td>( \tau(0) = 1 )</td>
</tr>
</tbody>
</table>

Clearly, \( \tau = (1, 1, 1, \ldots) \). For \( \sigma \), we compute

\[
\begin{align*}
\sigma &= \sigma(0) : \sigma' \\
&= 0 : (\sigma + \tau)
\end{align*}
\]
\[
= 0 : (\sigma + \tau)(0) : (\sigma + \tau)' \\
= 0 : (\sigma(0) + \tau(0)) : (\sigma' + \tau') \\
= 0 : 1 : (\sigma + 2\tau) \\
= 0 : 1 : (\sigma + 2\tau)(0) : (\sigma + 2\tau)' \\
= 0 : 1 : 2 : (\sigma + 3\tau) \\
\] and so on. We see: \(\sigma = (0, 1, 2, 3, \ldots)\).

**Example 3.21.** Let \(\sigma \in \mathbb{R}^\omega\) be defined by

<table>
<thead>
<tr>
<th>derivative</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sigma' = X \times \sigma)</td>
<td>(\sigma(0) = 1)</td>
</tr>
</tbody>
</table>

Then

\[
\sigma = \sigma(0) : \sigma' \\
= 1 : (X \times \sigma) \\
= 1 : (X \times \sigma)(0) : (X \times \sigma)' \\
= 1 : 0 : \sigma \\
= 1 : 0 : 1 : 0 : \sigma.
\]

Thus \(\sigma = (1, 0, 1, 0, 1, 0, \ldots)\).

**Exercise 3.22.** Compute the stream \(\sigma\) defined by

<table>
<thead>
<tr>
<th>derivative</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sigma' = X^2 \times \sigma)</td>
<td>(\sigma(0) = 1)</td>
</tr>
</tbody>
</table>

3.4. The operation of inverse

In Section 3.5, we shall solve linear equations in one unknown \(\tau\), such as

\[
\tau = 1 + (X \times \tau) \tag{3}
\]
(where, recall, \(1 = (1, 0, 0, 0, \ldots)\) by Notation 3.2).

**Exercise 3.23.** Here is a down-to-earth (but not very practical) way of solving Eq. (3). Let \(\tau = (\tau_0, \tau_1, \tau_2, \ldots)\) and substitute this expression on both sides of the equation. From this compute the value of \(\tau_0\), then that of \(\tau_1\), and so on.

Ideally, we would like to solve (3) by reasoning as follows:

\[
\tau = 1 + (X \times \tau), \\
\Rightarrow \tau - (X \times \tau) = 1, \\
\Rightarrow (1 - X) \times \tau = 1, \\
\Rightarrow \tau = \frac{1}{1 - X}.
\]
Recall, however, that we are not dealing with functions but with streams. Therefore it is by no means obvious what we mean by the ‘inverse’ of \((1 - X)\):

\[
\frac{1}{1 - X} = \frac{1}{(1, -1, 0, 0, 0, \ldots)} = ?
\]

Using stream differential equations, there turns out to be a very natural and convenient way to define the inverse of any stream \(\sigma\) with \(\sigma(0) \neq 0\). Here are a few examples first.

**Example 3.24.** The inverse \(\tau = \frac{1}{1 - X}\) of the stream \(1 - X\) should be such that \((1 - X) \times \tau = 1\). Equivalently, the stream \(\tau\) should satisfy

\[
\tau = 1 + (X \times \tau)
\]

(which is Eq. (3) that we started out with at the beginning of this section). A first observation is that this equation uniquely determines what the initial value of \(\tau\) should be

\[
\tau(0) = (1 + (X \times \tau))(0) \\
= 1 \quad \text{(using Corollary 3.14)}.
\]

Taking the stream derivative on both sides of the equation gives

\[
\tau' = (1 + (X \times \tau))' \\
= \tau \quad \text{(again using Corollary 3.14)}.
\]

We therefore see that \(\tau\) must satisfy the following stream differential equation:

\[
\begin{array}{|c|c|}
\hline
\text{derivative} & \text{initial value} \\
\tau' & 1 \\
\hline
\end{array}
\]

(One can also show that if \(\tau\) satisfies this differential equation, then \(\tau = 1 + (X \times \tau)\).) We have seen this type of differential equation before, and we know how to solve it. Namely, \(\tau = (1, 1, 1, \ldots)\) is its unique solution. Now we can define

\[
\frac{1}{1 - X} = (1, 1, 1, \ldots).
\]

**Exercise 3.25.** Check that \((1, 1, 1, \ldots) \times (1 - X) = 1\), using the definition of the product.

**Example 3.26.** The inverse \(\tau\) of the stream \(1 - X^2\) should satisfy \((1 - X^2) \times \tau = 1\), which is equivalent to

\[
\tau = 1 + (X^2 \times \tau).
\]

This gives \(\tau(0) = (1 + (X^2 \times \tau))(0) = 1\). For the derivative we find

\[
\tau' = (1 + (X^2 \times \tau))' \\
= X \times \tau \quad \text{(using Corollary 3.14)}.
\]
Thus $\tau$ is determined by the following equation:

<table>
<thead>
<tr>
<th>derivative $\tau'$</th>
<th>initial value $\tau(0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau' = X \times \tau$</td>
<td>$\tau(0) = 1$</td>
</tr>
</tbody>
</table>

This equation we recognize from Example 3.21, thus

$$\frac{1}{1 - X^2} = (1, 0, 1, 0, 1, 0, \ldots).$$

**Exercise 3.27.** Compute the inverse of the following streams:

(i) $1 + X$,
(ii) $1 - rX$ (for any $r \in \mathbb{R}$),
(iii) $(1 - X)^2$,
(iv) $(1, 1, 1, \ldots)$,
(v) $1 + X^2$.

**Exercise 3.28.** Prove, for all $r \in \mathbb{R}$,

$$\frac{r}{1 - X} = (r, r, r, \ldots).$$

Use coinduction to prove, for all $a, b \in \mathbb{R}$,

$$\text{zip} \left( \frac{a}{1 - X}, \frac{b}{1 - X} \right) = \frac{a + bX}{1 - X^2}.$$  

(The operation $\text{zip}$ was introduced in Example 2.9.)

A similar procedure works for any stream $\sigma$ with $\sigma(0) \neq 0$. From the requirement that

$$\sigma \times \frac{1}{\sigma} = 1$$  \hspace{1cm} (4)

one can deduce a stream differential equation as follows. Taking the initial value at both sides gives

$$\sigma(0) \times \frac{1}{\sigma(0)} = 1,$$

which implies $\frac{1}{\sigma}(0) = \frac{1}{\sigma(0)}$. Taking the derivative of both sides gives

$$\left( \sigma_0 \times \left( \frac{1}{\sigma} \right) \right)' + \left( \sigma' \times \frac{1}{\sigma} \right) = 0,$$

which implies

$$\left( \frac{1}{\sigma} \right)' = -\frac{1}{\sigma(0)} \times \sigma' \times \frac{1}{\sigma}.$$  

This leads to the following definition:
Definition 3.29. We define the inverse \( \frac{1}{\sigma} \) of a stream \( \sigma \) with \( \sigma(0) \neq 0 \) as the unique stream satisfying the following stream differential equation:

\[
\begin{align*}
(\frac{1}{\sigma})' &= -\frac{1}{\sigma(0)} \times \sigma' \times \frac{1}{\sigma} \\
(\frac{1}{\sigma})(0) &= \frac{1}{\sigma(0)}
\end{align*}
\]

We shall use the following notational convention: for all \( \sigma, \tau \in \mathbb{R}^\omega \) with \( \sigma(0) \neq 0 \),

\[
\tau \equiv \frac{1}{\sigma} \times \tau = \tau \times \frac{1}{\sigma}.
\]

This product is called the quotient of \( \tau \) and \( \sigma \).

As with sum and product, we can calculate with the operation of inverse in the same way as we compute with functions.

Proposition 3.30. For all \( \sigma, \tau \in \mathbb{R}^\omega \) with \( \sigma(0) \neq 0 \neq \tau(0) \),

\[
\begin{align*}
\sigma \times \frac{1}{\sigma} &= 1, \\
\frac{1}{\sigma} \times \tau &= \frac{1}{\sigma} \times \tau, \\
\frac{1}{\tau} &= \sigma.
\end{align*}
\]

Proof. The first equality follows by coinduction (Theorem 2.14) from the fact that the following relation:

\[
\left\{ \sigma \times \frac{1}{\sigma}, 1 \right\} \mid \sigma \in \mathbb{R}^\omega \}
\]

is a bisimulation relation. The second equation follows from the first one, and the last equation can again be proved by coinduction. (See [12] for details.) \( \square \)

Example 3.31.

\[
\begin{align*}
\frac{1}{(1 - X)^2} &= \frac{1}{(1 - X) \times (1 - X)} \\
&= \frac{1}{1 - X} \times \frac{1}{1 - X} \\
&= (1, 1, 1, \ldots) \times (1, 1, 1, \ldots) \\
&= (1, 2, 3, \ldots),
\end{align*}
\]

\[
\begin{align*}
\frac{1}{1 - X^2} &= \frac{1}{(1 - X) \times (1 + X)} \\
&= \frac{1}{1 - X} \times \frac{1}{1 + X} \\
&= (1, 1, 1, \ldots) \times (1, -1, 1, -1, 1, -1, \ldots) \\
&= (1, 0, 1, 0, 1, 0, \ldots).
\end{align*}
\]
We conclude this section with the following definition.

**Definition 3.32.** The product of a polynomial stream and the inverse of a polynomial stream is called a *rational stream*. Equivalently, a stream $\sigma$ is rational if there exist $n, m \geq 0$ and coefficients $r_0, \ldots, r_n, s_0, \ldots, s_m \in \mathbb{R}$ with $s_0 \neq 0$, such that

$$\sigma = \frac{r_0 + r_1 X + r_2 X^2 + \cdots + r_n X^n}{s_0 + s_1 X + s_2 X^2 + \cdots + s_m X^m}.$$

**Exercise 3.33.** Prove that the sum, the product, and the inverse of rational streams are again rational.

### 3.5. Solving linear equations

Next we show how to solve systems of linear equations. The solution of such systems can be computed in an algebraic manner, and will be expressed in terms of the constants and the operations of sum, product, and inverse. We shall only treat a few examples, which will be all that is needed later. (On the basis of these examples, however, it would not be very difficult to formulate and prove a more general result.)

**Example 3.34.** Consider the following system of equations:

$$\sigma = 1 + (X \times \tau),$$
$$\tau = X \times \sigma.$$

In order to find $\sigma$ and $\tau$, we compute as follows:

$$\sigma = 1 + (X \times \tau)$$
$$= 1 + (X \times X \times \sigma)$$
$$= 1 + (X^2 \times \sigma).$$

This implies

$$\sigma - (X^2 \times \sigma) = 1,$$
$$(1 - X^2) \times \sigma = 1.$$

Thus

$$\sigma = \frac{1}{1 - X^2},$$
$$\tau = \frac{X}{1 - X^2}.$$

**Example 3.35.** Consider the following system of equations:

$$\sigma = 1 + (X \times \tau),$$
$$\tau = 2 + (X \times \sigma).$$
In order to find $\sigma$ and $\tau$, we compute as before:

\[
\sigma = 1 + (X \times \tau) \\
= 1 + (X \times (2 + (X \times \sigma))) \\
= 1 + 2X + (X^2 \times \sigma).
\]

This implies

\[
\sigma - (X^2 \times \sigma) = 1 + 2X, \\
(1 - X^2) \times \sigma = 1 + 2X.
\]

Thus

\[
\sigma = \frac{1 + 2X}{1 - X^2}, \\
\tau = 2 + (X \times \sigma) \\
= 2 + \left( X \times \frac{1 + 2X}{1 - X^2} \right) \\
= \frac{2 - 2X^2}{1 - X^2} + \frac{X + 2X^2}{1 - X^2} \\
= \frac{2 + X}{1 - X^2}.
\]

**Exercise 3.36.** With $\sigma$ and $\tau$ as is Example 3.35, what is the value of $\sigma(n)$ and $\tau(n)$, for $n \geq 0$?

**Exercise 3.37.** Compute the solution of the following systems of equations:

(a) 
\[
\sigma = 1 + (2X \times \sigma) - (X \times \tau), \\
\tau = X \times \sigma,
\]

(b) 
\[
\sigma = 1 + (X \times \sigma) + (X \times \tau), \\
\tau = X \times \sigma.
\]

In conclusion of this section, we make the following general observation, the proof of which is omitted. The solutions of finite systems of linear equations such as the ones above, are always rational streams (cf. Definition 3.32). Conversely, any rational stream can be obtained as the solution of such a linear system of equations.

4. Stream circuits

Certain functions from $\mathbb{R}^\omega$ to $\mathbb{R}^\omega$ can be represented by means of graphical networks that are built from a small number of basic ingredients. Such networks can be viewed as implementations of stream functions. We call them stream circuits; in the literature, they are also referred to as (signal) flow graphs. Using the basic stream calculus from Section 3,
we shall give a formal but simple answer to the question precisely which stream functions can be implemented by such stream circuits.

4.1. Basic circuits

The circuits that we are about to describe, will generally have a number of input ends and a number of output ends. Here is an example of a simple circuit, consisting of one input and one output end:

\[ \rightarrow \]

The input end is denoted by the arrow shaft \( \rightarrow \) and the output end is denoted by the arrow head \( \Rightarrow \). For streams \( \sigma, \tau \in \mathbb{R}^\omega \), we shall write

\[ \sigma \Rightarrow \tau \]

and say that the circuit inputs the stream \( \sigma \) and outputs the stream \( \tau \). Writing the elements of these streams explicitly, this notation is equivalent to

\[ (\sigma_0, \sigma_1, \sigma_2, \ldots) \Rightarrow (\tau_0, \tau_1, \tau_2, \ldots), \]

which better expresses the intended operational behaviour of the circuit: It consists of an infinite sequence of actions, at time moments 0, 1, 2, \ldots. At each moment \( n \geq 0 \), the circuit simultaneously inputs the value \( \sigma_n \in \mathbb{R} \) at its input end and outputs the value \( \tau_n \in \mathbb{R} \) at its output end. In general, this value \( \tau_n \) depends both on the value \( \sigma_n \) and on the values \( \sigma_i \) that have been taken as inputs at earlier time moments \( i < n \). Note that this implies that circuits have memory.

Next we present the four basic types of circuits, out of which all other circuits in this section will be constructed.

(a) For every \( a \in \mathbb{R} \), we define a circuit with one input and one output end, called an \( a \)-multiplier, for all \( \sigma, \tau \in \mathbb{R}^\omega \), by

\[ \sigma \Rightarrow \tau \quad \Leftrightarrow \quad \tau_n = a \cdot \sigma_n \quad \text{all } n \geq 0 \]

\[ \Leftrightarrow \quad \tau = a \times \sigma. \]

This circuit takes, at any moment \( n \geq 0 \), a value \( \sigma_n \) at its input end, multiplies it with the constant \( a \), and outputs the result \( \tau_n = a \cdot \sigma_n \) at its output end. It defines, in other words, a function that assigns to an input stream \( \sigma \) the output stream \( \tau = a \times \sigma \).

Occasionally, it will be more convenient to write the multiplying factor \( a \) as a superscript or subscript of the arrow:

\[ a \Rightarrow \equiv \quad a \Rightarrow \equiv \quad a \Rightarrow \]

\[ a \Rightarrow \equiv \quad a \Rightarrow \equiv \quad a \Rightarrow \]
(b) The *adder* circuit has two input and one output ends, and is defined, for all \( \sigma, \tau, \rho \in \mathbb{R}^\omega \), by

\[
\sigma + \rho \quad \iff \quad \rho_n = \sigma_n + \tau_n \quad \forall n \geq 0
\]

At moment \( n \geq 0 \), the adder simultaneously inputs the values \( \sigma_n \) and \( \tau_n \) at its input ends, and outputs their sum \( \rho_n = \sigma_n + \tau_n \) at its output end.

(c) The *copier* circuit has one input and two output ends and is defined, for all \( \sigma, \tau, \rho \in \mathbb{R}^\omega \), by

\[
\sigma c \tau \quad \iff \quad \tau_n = \sigma_n = \rho_n \quad \forall n \geq 0
\]

At any moment \( n \geq 0 \), the copier inputs the value \( \sigma_n \) at its input end, and outputs two identical copies \( \tau_n = \sigma_n = \rho_n \) at its output ends.

(d) A *register* circuit has one input and one output end and is defined, for all \( \sigma, \tau \in \mathbb{R}^\omega \), by

\[
\sigma \downarrow R \tau \quad \iff \quad \tau_0 = 0 \quad \text{and} \quad \tau_n = \sigma_{n-1} \quad \forall n \geq 1
\]

\[
\iff \quad \tau = (0, \sigma_0, \sigma_1, \sigma_2, \ldots).
\]

The register circuit can be viewed as consisting of a one-place memory cell that initially contains the value 0. The register starts its activity, at time moment 0, by outputting its value \( \tau_0 = 0 \) at its output end, while it simultaneously inputs the value \( \sigma_0 \) at its input end, which is stored in the memory cell. At any future time moment \( n \geq 1 \), the value \( \tau_n = \sigma_{n-1} \) is output and the value \( \sigma_n \) is input and stored. (For obvious reasons, the register circuit is sometimes also called a *unit delay*.) Recalling the constant stream \( X = (0, 1, 0, 0, 0, \ldots) \) from Definition 3.6, it is an immediate consequence of Proposition 3.7 that we have, for all \( \sigma, \tau \in \mathbb{R}^\omega \),

\[
\sigma \downarrow R \tau \quad \iff \quad \tau = X \times \sigma.
\]

4.2. Circuit composition

We can construct a larger circuit out of two smaller ones by connecting output ends of the first to input ends of the second. Rather than giving a fully general and formal definition, which is not very difficult but a bit tedious, we prefer to explain circuit composition by means of a number of examples. These will be sufficiently representative to teach the reader how to construct his or her own circuits. We shall mostly concentrate on circuits with only one input and one output end, but nothing of what follows depends on such a restriction.

**Example 4.1.** For the composition of a 2-multiplier and a 3-multiplier, we shall write

\[
\downarrow 2 \circ \downarrow 3
\]

We call the connection point \( \circ \) an (internal) *node* of the composed circuit. A computation step of this circuit, at any moment in time, consists of the simultaneous occurrence of the
following actions: a value is input at the input end of the 2-register; it is multiplied by 2 and output at the output end of the 2-register; the result is input at the input end of the 3-register, is multiplied by 3 and is output at the output end of the 3-multiplier. More formally, and fortunately also more succinctly, we define the behaviour of the composed circuit, for all \( \sigma, \tau \in \mathbb{R}^\omega \), by

\[
\sigma \xrightarrow{2} \circ \xrightarrow{3} \tau \\
\iff \sigma \xrightarrow{2} \exists \rho \xrightarrow{3} \tau \\
\iff \exists \rho \in \mathbb{R}^\omega : \sigma \xrightarrow{2} \rho \quad \text{and} \quad \rho \xrightarrow{3} \tau.
\]

We shall consider all three of the above notations as equivalent. Combining the definitions of a 2- and 3-multiplier, we can in the above example easily compute how the output stream \( \tau \) depends on the input stream \( \sigma \):

\[
\sigma \xrightarrow{2} \circ \xrightarrow{3} \tau \\
\iff \exists \rho \in \mathbb{R}^\omega : \sigma \xrightarrow{2} \rho \quad \text{and} \quad \rho \xrightarrow{3} \tau \\
\iff \exists \rho \in \mathbb{R}^\omega : \rho = 2 \times \sigma \quad \text{and} \quad \tau = 3 \times \rho \\
\iff \tau = 6 \times \sigma.
\]

Note that the stream \( \rho \) is uniquely determined by the stream \( \sigma \). The motivation for our notation “\( \exists \rho \)” is not so much to suggest that there might be more possible candidate streams for \( \rho \), but rather to emphasise the fact that in order to express the output stream \( \tau \) in terms of \( \sigma \), we have to compute the value of the stream \( \rho \) in the middle.

**Example 4.2.** We can compose circuits, more generally, with several output ends with circuits having a corresponding number of input ends, as in the following example:

\[
\begin{align*}
\sigma \xrightarrow{c} + & \xrightarrow{\circ} \tau \\
\iff \exists \gamma, \delta \in \mathbb{R}^\omega : \sigma \xrightarrow{c} \gamma & \quad \text{and} \quad \delta \xrightarrow{\circ} \tau \\
\iff \exists \gamma, \delta \in \mathbb{R}^\omega : \sigma = \gamma = \delta & \quad \text{and} \quad \tau = \gamma + \delta \\
\iff \tau = 2 \times \sigma.
\end{align*}
\]
Exercise 4.3. (a) Prove that the following two circuits are equivalent, for all $a, b \in \mathbb{R}$:

\[
\begin{array}{c}
\downarrow a \to o \uparrow b \equiv \downarrow a \cdot b \\
\end{array}
\]

(b) Construct a number of different circuits that are all equivalent to a 1-multiplier. How many such circuits do there exist?

It will be convenient to have adders with more than two inputs and, similarly, copiers with more than two outputs.

Definition 4.4. We define a ternary adder as the composition of two binary adders as follows:

For input streams $\sigma, \tau, \rho \in \mathbb{R}^D$, it produces the output stream $\sigma + \tau + \rho$. We define a ternary copier by the following composition:

It takes one input stream and produces three identical copies as output streams. Adders and copiers with four or more inputs and outputs can be constructed in a similar fashion.

Example 4.5. The following circuit combines (various instances of) all four basic circuit types:

In order to express the output stream $\tau$ for a given input stream $\sigma$, we have to compute one intermediate stream for each of the (nine) internal nodes $o$ in the circuit above.
In other words, we have to compute \( \rho_1, \ldots, \rho_9 \in \mathbb{R}^\omega \) such that

\[
\begin{array}{c}
\sigma \downarrow 1 \rightarrow 2 \rightarrow \rho_1 \\
\sigma \downarrow 3 \rightarrow 3 \rightarrow \rho_3 \\
\sigma \downarrow 4 \rightarrow R \rightarrow \rho_4 \\
\rho_6 \downarrow 5 \rightarrow \rho_7 \rightarrow R \rightarrow \rho_8 \\
\rho_8 \downarrow 6 \rightarrow R \rightarrow \rho_9 \\
\end{array}
\]

Using the definitions of the basic circuits, and computing from left to right, we find:

\[
\begin{array}{c}
\sigma \downarrow 2 \rightarrow 2 \rightarrow \sigma \\
\sigma \downarrow 3 \rightarrow 3 \rightarrow 3\sigma \rightarrow R \rightarrow 3\sigma \\
\sigma \downarrow 7 \rightarrow -7\sigma \rightarrow R \rightarrow -7\sigma \rightarrow R \rightarrow -7\sigma^2 \\
\end{array}
\]

(To save space, we have omitted the symbol \( \times \) for multiplication.) We can now express the output stream \( \tau \) in terms of the input stream \( \sigma \) as follows:

\[
\tau = (2 \times \sigma) + (3X \times \sigma) + (-7X^2 \times \sigma)
\]

\[
= (2 + 3X - 7X^2) \times \sigma.
\]

The circuit above computes, in other words, the following function on streams:

\[
f : \mathbb{R}^\omega \rightarrow \mathbb{R}^\omega, \quad f(\sigma) = (2 + 3X - 7X^2) \times \sigma.
\]

If we supply the circuit with the input stream \( \sigma = 1 \) (= (1, 0, 0, 0, …)) then the output stream is

\[
\tau = f(1)
\]

\[
= 2 + 3X - 7X^2.
\]

**Definition 4.6.** Let \( f : \mathbb{R}^\omega \rightarrow \mathbb{R}^\omega \) be a stream function.

(a) If a circuit with one input end and one output end transforms every input stream \( \sigma \in \mathbb{R}^\omega \) to an output stream \( \tau = f(\sigma) \), then we say that the circuit implements (or: is an implementation of) the function \( f \).

(b) We call the output stream \( \tau = f(1) \), obtained on input \( \sigma = 1 \), the stream generated by the circuit.

**Convention 4.7.** In order to reduce the size of the diagrams with which we depict stream circuits, it will often be convenient to leave the operations of copying and addition implicit. In this manner, we can, for instance, draw the circuit of Example 4.5 above as follows:
The (respective elements of the) stream $\sigma$ gets copied along each of the three outgoing arrows. Similarly, the stream $\tau$ will be equal to the sum of the output streams of the three incoming arrows. This convention saves a lot of writing. Moreover, if we want to express $\tau$ in terms of $\sigma$, we now have only three internal streams to compute. If a node has both incoming and outgoing arrows, such as

then first the values of the output streams of the incoming arrows have to be added; then the resulting sum is copied and given as an input stream to each of the outgoing arrows. Consider for instance the circuit below. It has input streams $\sigma$ and $\tau$, an intermediate stream $\gamma$, and output streams $\delta$ and $\varepsilon$ in $\mathbb{R}^\omega$:

satisfying

$$
\begin{align*}
\gamma &= 2\sigma + (X \times \tau) \\
\delta &= X \times \gamma \\
&= (2X \times \sigma) + (X^2 \times \tau) \\
\varepsilon &= S\gamma \\
&= 10\sigma + (5X \times \tau).
\end{align*}
$$

**Example 4.8.** Consider the following circuit:

(Note that this circuit contains one copier and two adders, which are invisible because we are using Convention 4.7.) In order to express $\tau$ in terms of $\sigma$, we have to compute two intermediate streams $\gamma$ and $\delta$ such that:

$$
\begin{align*}
\gamma &= 2\sigma + (X \times \tau) \\
\delta &= X \times \gamma \\
&= (2X \times \sigma) + (X^2 \times \tau) \\
\varepsilon &= S\gamma \\
&= 10\sigma + (5X \times \tau).
\end{align*}
$$
It follows that
\[ \gamma = -7 \times \sigma, \]
\[ \delta = (3 \times \sigma) + (X \times \gamma) = (3 \times \sigma) + (-7X \times \sigma) \]
\[ \tau = (2 \times \sigma) + (X \times \delta) = (2 \times \sigma) + X \times ((3 \times \sigma) + (-7X \times \sigma)) = (2 \times \sigma) + (3X \times \sigma) + (-7X^2 \times \sigma) = (2 + 3X - 7X^2) \times \sigma. \]

We see that this circuit has the same stream function as the circuit of Example 4.5. An important difference is that the latter circuit contained three register circuits, whereas the present circuit only uses two. Thus it uses less memory.

**Example 4.9.** We compute the stream function implemented by the following circuit, with input stream \( \sigma \), output stream \( \tau \), and intermediate streams \( \gamma \) and \( \delta \):

\[
\begin{array}{c}
\sigma \\
\downarrow \\
\gamma \\
\downarrow \downarrow \\
R \\
\downarrow \downarrow \\
\delta \\
\downarrow \\
\tau \\
\downarrow \\
\sigma
\end{array}
\]

We have:
\[ \gamma = X \times \sigma, \]
\[ \delta = (3 \times \sigma) + (X^2 \times \sigma), \]
\[ \tau = (5 \times \gamma) + (X \times \delta) = (8X + X^3) \times \sigma. \]

Thus the stream function implemented by this circuit is \( f : \mathbb{R}^{\sigma} \rightarrow \mathbb{R}^{\sigma} \) with \( f(\sigma) = (8X + X^3) \times \sigma \), for all \( \sigma \in \mathbb{R}^{\sigma} \). An equivalent circuit, implementing the same stream function, is given by

\[
\begin{array}{c}
S \\
\rightarrow \\
1 \\
\rightarrow \\
R \\
\rightarrow \\
\rightarrow \\
R \\
\rightarrow \\
R \\
\rightarrow \\
R
\end{array}
\]

**Exercise 4.10.** Show that the two circuits of Example 4.9 both implement the same stream function.

**Exercise 4.11.** (a) Draw a picture of the following circuit showing explicitly the adders and copiers that are used:

\[
\begin{array}{c}
\sigma \\
\downarrow \\
S \\
\downarrow \downarrow \\
R \\
\downarrow \downarrow \\
R \\
\downarrow \downarrow \\
R \\
\downarrow \downarrow \\
R \\
\downarrow \downarrow \\
T
\end{array}
\]

Then compute the stream function \( f : \mathbb{R}^{\sigma} \rightarrow \mathbb{R}^{\sigma} \) that is implemented by this circuit.
(b) Compute the stream function \( f : \mathbb{R}_o^o \rightarrow \mathbb{R}_o^o \) that is implemented by the following circuit:

(c) Compute the stream function that is implemented by the following circuit:

The following proposition characterizes which stream functions can be implemented by the type of circuits that we have been considering so far.

**Proposition 4.12.** For all \( n \geq 0 \) and \( r_0, \ldots, r_n \in \mathbb{R} \), each of the following two circuits:

implies the stream function \( f : \mathbb{R}_o^o \rightarrow \mathbb{R}_o^o \) given, for all \( \sigma \in \mathbb{R}_o^o \), by

\[
f(\sigma) = \rho \times \sigma
\]

where the stream \( \rho \) (generated by these circuits) is the polynomial

\[
\rho = r_0 + r_1 X + r_2 X^2 + \cdots + r_{n-1} X^{n-1} + r_n X^n.
\]

**Exercise 4.13.** Prove Proposition 4.12.

**Exercise 4.14.** (a) Give a circuit that generates the polynomial stream \( 1 - 3X + 5X^2 \). Same question for \( 1 + X \).

(b) What is the stream function computed by the composition of these two circuits?

**Exercise 4.15.** (a) Consider the function \( f : \mathbb{R}_o^o \rightarrow \mathbb{R}_o^o \) that assigns to every \( \sigma \in \mathbb{R}_o^o \) the stream \( f(\sigma) = \tau \) defined by \( \tau_0 = \sigma_0 \) and, for all \( n \geq 1 \), by

\[
\tau_n = \sigma_{n-1} + \sigma_n.
\]

Construct a circuit that implements the function \( f \).
(b) Can you do the same in case
\[ \tau_n = \sigma_0 + \sigma_1 + \cdots + \sigma_n \]
for all \( n \geq 0 \)?

**Exercise 4.16.** Take any of the circuits that we have seen so far. (In fact, take any circuit.) Reverse all the arrows, replace all adders by copiers, and replace all copiers by adders. Then show that both the original and the reversed circuit compute the same stream function.

### 4.3. Circuits with feedback loops

The use of feedback loops in stream circuits increases their expressive power substantially. We shall again start with a few examples and then give a simple and precise characterization of all stream functions that can be implemented by circuits with feedback loops.

**Example 4.17.** Here is the simplest example of a circuit with feedback:

![Circuit Diagram](image)

In spite of its simplicity, this circuit is already quite interesting. Before we give a formal computation of the stream function that this circuit implements, we give an informal description of its behaviour first. Assuming that we have an input stream \( \sigma = (\sigma_0, \sigma_1, \sigma_2, \ldots) \), we compute the respective elements of the output stream \( \tau = (\tau_0, \tau_1, \tau_2, \ldots) \). Recall that a register can be viewed as a one-place memory cell with initial value 0. At moment 0, our circuit begins its activity by inputting the first value \( \sigma_0 \) at its input end. The present value of the register, 0, is added to this and the result \( \tau_0 = \sigma_0 + 0 = \sigma_0 \) is the first value to be output. At the same time, this value \( \sigma_0 \) is copied and stored as the new value of the register.

The next step consists of inputting the value \( \sigma_1 \), adding the present value of the register, \( \sigma_0 \), to it, and outputting the resulting value \( \tau_1 = \sigma_0 + \sigma_1 \). At the same time, this value \( \sigma_0 + \sigma_1 \) is copied and stored as the new value of the register. The next step will input \( \sigma_2 \) and output the value \( \tau_2 = \sigma_0 + \sigma_1 + \sigma_2 \). And so on. We find:

\[ \tau = (\sigma_0, \sigma_0 + \sigma_1, \sigma_0 + \sigma_1 + \sigma_2, \ldots). \]

Next we show how the same answer can be obtained, more formally and more systematically, by applying a bit of basic stream calculus. As before, we try to express the output stream \( \tau \) in terms of the input stream \( \sigma \) by computing the values of intermediate streams \( \rho_1, \rho_2, \rho_3 \in \mathbb{R}^\omega \), corresponding to the three internal nodes of the circuit, such that

![Circuit Diagram](image)

Note that the values of \( \rho_1, \rho_2, \rho_3 \) are mutually dependent because of the presence of the feedback loop: \( \rho_3 \) depends on \( \rho_1 \) which depends on \( \rho_2 \) which depends on \( \rho_3 \). Fortunately,
we have used coinduction in Section 3 to develop a stream calculus that is precisely fit for this type of circularity. Unfolding the definitions of the basic circuits of which the above circuit is composed (one adder, one register, and one copier), we find the following system of equations:

\[
\begin{align*}
\rho_1 &= X \times \rho_2, \\
\rho_3 &= \sigma + \rho_1, \\
\rho_2 &= \rho_3, \\
\tau &= \rho_3.
\end{align*}
\]

We have learned in Section 3.5 how to solve such a system of equations:

\[
\tau = \rho_3 = \sigma + \rho_1 = \sigma + (X \times \rho_2) = \sigma + (X \times \tau).
\]

As a consequence, \( \tau - (X \times \tau) = \sigma \), which is equivalent to \( \tau = \frac{1}{1-X} \times \sigma \). Thus the stream function \( f : \mathbb{R}^\omega \to \mathbb{R}^\omega \) that is implemented by the feedback circuit is given, for all \( \sigma \in \mathbb{R}^\omega \), by

\[
f(\sigma) = \frac{1}{1-X} \times \sigma.
\]

Somewhat surprisingly, maybe, we see that this function consists again of the convolution product of the argument \( \sigma \) and a constant stream \( \frac{1}{1-X} \). The main difference with the examples in the previous sections is that in the present example this constant stream is no longer a polynomial stream, but the inverse of a polynomial stream.

**Exercise 4.18.** We still have to check that the first informal and the second formal computation of the function implemented by the feedback circuit in Example 4.17 coincide. Prove, for all \( \sigma \in \mathbb{R}^\omega \), that

\[
\frac{1}{1-X} \times \sigma = (\sigma_0, \sigma_0 + \sigma_1, \sigma_0 + \sigma_1 + \sigma_2, \ldots).
\]

Not every feedback loop gives rise to a circuit with a well-defined behaviour. Consider for instance the following circuit, with input stream \( \sigma \), output stream \( \tau \), and internal streams \( \rho_1, \rho_2, \rho_3 \):

\[
\begin{array}{c}
\rho_2 \\
\sigma \\
\rho_3 \\
\tau
\end{array} \quad \begin{array}{c}
\leftarrow \quad \\
\quad \quad \\
\quad \\
\rightarrow
\end{array}
\]

In this circuit, we have replaced the register feedback loop of Example 4.17 by a 1-multiplier. If we try to compute the stream function of this circuit as before, we find the following system
of equations:
\[
\begin{align*}
\rho_1 & = 1 \times \rho_2, \\
\rho_3 & = \sigma + \rho_1, \\
\rho_2 & = \rho_3, \\
\tau & = \rho_3.
\end{align*}
\]
This leads to \( \rho_2 = \sigma + \rho_3 \), which implies \( \sigma = 0 \). But \( \sigma \) is supposed to be an arbitrary input stream, so this does not make sense.

Problems like this can be avoided by assuming that circuits have the following property.

**Assumption 4.19.** From now on, we shall consider only circuits in which every feedback loop passes through at least one register circuit.

This formulation is admittedly informal, but as we shall see, it is precise enough for our present purposes. (Moreover, it can be made completely formal without too much effort.) Under Assumption 4.19, all circuits will have a well-defined behaviour.

**Exercise 4.20.** What happens if we replace the register circuit of Example 4.17 by a 2-multiplier (instead of the 1-multiplier in the example above)?

**Example 4.21.** Consider the following circuit:

\[
\begin{array}{c}
\circ \leftarrow \circ \leftarrow \circ \leftarrow R \leftarrow \circ \\
\circ \leftarrow \circ \leftarrow \circ \leftarrow \circ \leftarrow \circ
\end{array}
\]

Taking an input stream \( \sigma \) and output stream \( \tau \) and computing the values of the internal streams, we find

\[
\begin{array}{c}
2X \tau \leftarrow 2X \tau \leftarrow R \leftarrow \tau \\
\sigma \leftarrow \tau \leftarrow \tau \leftarrow c \leftarrow \tau
\end{array}
\]

It follows that \( \tau = \sigma + (2X \times \tau) \), thus

\[
\tau = \frac{1}{1 - 2X} \times \sigma.
\]

Taking \( \sigma = 1 \), we see that the stream generated by this circuit is

\[
\frac{1}{1 - 2X} = (1, 2, 2^2, 2^3, \ldots).
\]

**Exercise 4.22.** (a) For every \( a \in \mathbb{R} \), compute the stream function implemented by the following circuit:

\[
\begin{array}{c}
\circ \leftarrow a \leftarrow \circ \leftarrow R \leftarrow \circ \\
\circ \leftarrow \circ \leftarrow \circ \leftarrow \circ
\end{array}
\]
(b) The same question for

```
   o ← R → o ← R → o
  ┌───────────┐
  │           │
  │           │
  └───────────┘
```

Example 4.23. Consider the following circuit:

```
   o ← R → o ← R → o
  ┌───────────┐
  │           │
  │           │
  └───────────┘
```

Using (a variation of) Convention 4.7, we may omit all adders and copiers and use the following equivalent diagram:

```
   o ← R → o ← R → o
  ┌───┐
  │ 1 │
  │   │
  │   │
  └───┘
```

One has to be a bit careful, though. Taking an input stream $\sigma$, an output stream $\tau$, and internal streams $\gamma$ and $\delta$, we get

```
   σ₁ → γ → Γ₁ → δ → τ
```

The tricky part is now to realise that, for instance, $\gamma$ equals the sum of all the incoming arrows, which are two: one from $\sigma$ and one from $\gamma$ itself. Note that $\gamma$ is copied along both outgoing arrows, including the one to itself. As a consequence, we get $\gamma = \sigma + (X \times \gamma)$. For $\delta$, we have something similar. This leads to the following system of equations:

\[
\begin{align*}
\gamma &= \sigma + (X \times \gamma), \\
\delta &= \gamma + (X \times \delta), \\
\tau &= \delta.
\end{align*}
\]

Using $\gamma = \frac{1}{1-X} \times \sigma$ and $\delta = \frac{1}{1-X} \times \gamma$, it then follows that

\[
\begin{align*}
\tau &= \delta \\
&= \frac{1}{1-X} \times \gamma \\
&= \frac{1}{1-X} \times \frac{1}{1-X} \times \sigma \\
&= \frac{1}{(1-X)^2} \times \sigma.
\end{align*}
\]

The stream generated by this circuit, obtained by taking $\sigma = 1$, is

\[
\frac{1}{(1-X)^2} = (1, 2, 3, \ldots).
\]
Exercise 4.24. Compute the stream function implemented by the following circuit:

4.4. Circuits and rational streams

Next we present the main result of the present paper. It is a characterization of which stream functions can be implemented by finite stream circuits. We formulate it for finite circuits that have one input and one output end, but it can be easily generalised to circuits with many inputs and outputs.

Theorem 4.25. (a) Let $C$ be any finite stream circuit, possibly containing feedback loops (that always pass through at least one register). The stream function $f : \mathbb{R}^\omega \to \mathbb{R}^\omega$ implemented by $C$ is always of the form:

$$f(\sigma) = \rho \times \sigma$$

for all $\sigma \in \mathbb{R}^\omega$ and for some fixed rational stream

$$\rho = \frac{r_0 + r_1X + r_2X^2 + \cdots + r_nX^n}{s_0 + s_1X + s_2X^2 + \cdots + s_mX^m}$$

with $n, m \geq 0$, $r_0, \ldots, r_n, s_0, \ldots, s_m \in \mathbb{R}$, and $s_0 \neq 0$.

(b) Let $f : \mathbb{R}^\omega \to \mathbb{R}^\omega$ be a stream function of the form, for all $\sigma \in \mathbb{R}^\omega$:

$$f(\sigma) = \rho \times \sigma$$

for some fixed rational stream $\rho$. Then there exists a finite stream circuit $C$ that implements $f$.

Proof. We have seen many examples in the previous sections bearing witness to statement (a). For a general proof, consider a finite circuit $C$ containing $k \geq 1$ registers. We associate with the input end of $C$ a stream $\sigma$ and with the output end of $C$ a stream $\tau$. With the output end of each register $R_i$, we associate a stream $\alpha_i$. For the input end of each register $R_i$, we look at all incoming paths that: (i) start in either an output end of any of the registers or the input end of $C$, (ii) lead via adders, copiers, and multipliers, (iii) to the input end of $R_i$. Because of Assumption 4.19, there are only finitely many of such paths. This leads to an equation of the form

$$\alpha_i = (a_i^1 \times X \times \alpha_i^1) + \cdots + (a_i^k \times X \times \alpha_i^k) + (a_i \times X \times \sigma)$$

for some $a_i, a_i^j \in \mathbb{R}$. We have one such equation for each $1 \leq i \leq k$. Solving this system of $k$ equations in stream calculus as before, yields for each register an expression $\alpha_i = \rho_i \times \sigma$,
for some rational stream \( \rho_i \). Finally, we play the same game for \( \tau \), at the output end of \( C \), as we did for each of the registers. This will yield the following type of expression for \( \tau \):

\[
\tau = (b_1 \times \alpha_1) + \cdots + (b_k \times \alpha_k) + (b \times \sigma) \\
= ((b_1 \times \rho_1) + \cdots + (b_k \times \rho_k) + b) \times \sigma
\]

for some \( b, b_i \in \mathbb{R} \), which proves (a). For (b), we treat only the special case that

\[
\rho = \frac{r_0 + r_1 X + r_2 X^2 + r_3 X^3}{1 + s_1 X + s_2 X^2 + s_3 X^3},
\]

where we have taken \( n = m = 3 \) and \( s_0 = 1 \). The general case is not more difficult, just more writing. We claim that the following circuit implements the function \( f(\sigma) = \rho \times \sigma \) (all \( \sigma \in \mathbb{R}^{\omega} \)):

\[
\begin{array}{c}
\sigma_1 \\
\rightarrow \\
1 \\
\rightarrow \\
\rho_0 \\
\rightarrow \\
\rho_1 \\
\rightarrow \\
\rho_2 \\
\rightarrow \\
\rho_3 \\
\rightarrow \\
\tau
\end{array}
\]

where we have denoted input and output streams by \( \sigma \) and \( \tau \), and intermediate streams by \( \rho_0, \rho_1, \rho_2, \rho_3 \). They satisfy the following equations:

\[
\begin{align*}
\rho_0 &= \sigma - (s_1 \times \rho_1) - (s_2 \times \rho_2) - (s_3 \times \rho_3), \\
\rho_1 &= X \times \rho_0 , \\
\rho_2 &= X \times \rho_1 , \\
\rho_3 &= X \times \rho_2 , \\
\tau &= (r_0 \times \rho_0) + (r_1 \times \rho_1) + (r_2 \times \rho_2) + (r_3 \times \rho_3).
\end{align*}
\]

It follows that

\[
\rho_0 = \sigma - (s_1 X \times \rho_0) - (s_2 X^2 \times \rho_0) - (s_3 X^3 \times \rho_0).
\]

As a consequence, we have, for \( i = 0, 1, 2, 3 \), that

\[
\rho_i = \frac{X^i}{1 + s_1 X + s_2 X^2 + s_3 X^3} \times \sigma.
\]

This implies

\[
\tau = \frac{r_0 + r_1 X + r_2 X^2 + r_3 X^3}{1 + s_1 X + s_2 X^2 + s_3 X^3} \times \sigma,
\]

whereby the claim above is proved. \( \square \)

**Corollary 4.26.** A stream \( \rho \in \mathbb{R}^{\omega} \) is rational if and only if it is generated by a (finite) stream circuit.
Exercise 4.27. Give for each of the streams below a circuit that generates it:

(i) \[
\frac{1 + X + X^2}{1 - X}.
\]

(ii) \[
\frac{X + X^3 + X^5}{1 + X^2 + X^4 + X^6}.
\]

(iii) \[
\frac{X^3}{1 - X - X^2}.
\]

5. Discussion and related work

In the language of category theory, any stream automaton (over the alphabet \(A\)) is a coalgebra of the functor that maps a set \(S\) to the set \(A \times S\). The set of streams is a final coalgebra of this functor. The set of streams is just one example of an interesting final coalgebra. For many more examples and some of the basic elements of the theory of ‘universal’ coalgebra, see [11]. The notion of bisimulation is due to Park and Milner [10,9], who designed it as a notion of equivalence for a theory of concurrent processes. (It existed already before, under the name of \(p\)-relation, in the world of Kripke frames and modal logic [14].) Final coalgebras have been used as models for many dynamical systems at least since [4,8]. It was not until a categorical generalisation of the notion of bisimulation was introduced, by Aczel and Mendler [1], that coinduction (both as a definition and as a proof principle) was taken more seriously. By now, there is a host of literature on many aspects of both theory and applications of coalgebra. Rather than trying to give an overview of the relevant literature here, we refer the reader to the proceedings of the annual international workshops on Coalgebraic Methods in Computer Science (CMCS), which started in 1998 (see [5]). More information on stream calculus can be found in [12].

As we mentioned in the introduction, the characterisation of finite circuits in terms of rational streams, in Theorem 4.25, is well known in the world of signal processing and linear systems; see, for instance, [7, p. 694]. The present approach is conceptually simpler, using only streams and coinduction and without the need to refer to \(Z\)-transforms, and can thereby be more easily generalised to other situations. For instance, such generalisations have recently [3] been applied to \(Reo\), a calculus developed by Farhad Arbab [2] for the compositional construction of component connectors. Another example is the use of \(bitstreams\), infinite sequences of 0’s and 1’s, in the semantics of digital circuits, in [13].

References


