Software ENgineering

Algebra, bitstreams, and circuits

J.J.M.M. Rutten

REPORT SEN-R0502 JANUARY 2005

ISSN 1386-369X

Algebra, bitstreams, and circuits

ABSTRACT
We define, analyse, and relate in a uniform way four different algebraic structures on the set of bitstreams, motivating each of them in terms of the digital circuits they can describe. For one of these, the 2-adic numbers, we characterise a class of linear digital circuits in terms of rational streams.

2000 Mathematics Subject Classification: 68Q10,45,55,70,85
Keywords and Phrases: Coalgebra, stream calculus, digital circuit, coinduction
Note: To appear in volume 16 of the series ``Contributions to General Algebra'', Verlag Johannes Heyn, Klagenfurt, 2005.

Algebra, Bitstreams, and Circuits

J.J.M.M. Rutten
CWI and VUA
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Email: janr@cwi.nl, URL: www.cwi.nl/∼janr

January 13, 2005

Abstract: We define, analyse, and relate in a uniform way four different algebraic structures on the set of bitstreams (infinite sequences of 0's and 1's), motivating each of them in terms of the digital circuits they can describe. For one of these, the 2-adic numbers, we characterise a class of linear digital circuits in terms of rational streams.

Note: To appear in volume 16 of the series "Contributions to General Algebra", Verlag Johannes Heyn, Klagenfurt, 2005.

1 Introduction

We study the set 2^ω of bitstreams: infinite sequences of 0's and 1's. Exploiting the fact that 2^ω carries a final coalgebra structure given by the operation of (stream) *derivative* [Brz64], we define in a uniform manner various operators on 2^ω by means of (stream) differential equations, and prove various properties about them by *coinduction* (cf. [JR97]). Next we use bitstreams and the various bitstream operators to define the semantics of *sequential* digital circuits, which have memory and allow feedback, and will include examples such as flip-flops, half- and full adders. Circuits without feedback loops and memory (sometimes called *logical* circuits) can be described by means of functions from Booleans to Booleans, which have been broadly studied and are well understood [Weg87]. Here we use (functions on) *streams* of Booleans instead of just Booleans, because output values may depend on previous input values, due to the presence of feedback and memory.

The operators on 2^ω come in four groups, each of which constitutes a different algebraic structure on 2^ω: a Boolean algebra, a Kleene algebra [Koz94], and two integral domains including that of the 2-adic integers [Gou93]. What will be new here are not these structures as such, which are well-known (with the possible exception of the bitstream Kleene algebra), but the uniform way in which the operators of these algebras are defined (or, if you like, characterised), by means of stream differential equations. This will allow us to reason about mixed algebraic expressions, containing operators from different algebraic structures, and to prove so-called *mixed laws* by coinduction. These mixed expressions will come up naturally in the semantics of sequential circuits. Moreover, we shall give a complete characterisation of the class of *2-adic linear* circuits, which are built using a basic gate for 2-adic sum, in terms of rational bitstreams.

The main contributions of the present paper are: (a) the uniform definition by means of stream differential equations of all the operators of all four algebraic structures; (b) the formulation and systematic proof (by coinduction) of various mixed laws; (c) the introduction and study of the Kleene algebra structure on 2^ω, which turns out to be a basic and useful model for sequential circuits; and (d) the complete characterisation of a class of linear circuits in terms of rational streams.

There does not seem to exist much literature on the systematic study of bitstream algebra in the context of digital circuits. Exceptions are the work by Vuillemin [Vui94, Vui00, Vui03], who analyses three of the four algebraic structures mentioned above (not the Kleene algebra), and studies algorithms for the construction of digital circuits from algebraic descriptions. Apart from the latter point, which we do not address here, the present treatment of bitstreams extends the results of Vuillemin by points (a) through (d) above. Bitstreams or *binary sequences* are also prominent in the study of linear feedback shift registers [Gol82]. Some algebra is used there: bitstreams are typically represented as formal power series with coefficients in the integers modulo 2. However, operations for multiplication do not seem to play an important role in that theory, and mixed algebraic expressions do not occur at all. The 2-adic numbers have been used in a generalisation of shift registers called feedback shift registers with carry operation [GK97]; mixed algebra does not play a role there either.

2 The final coalgebra of bitstreams

We introduce the set of bitstreams, the operations of initial value and stream derivative, and the proof and definition principles of coinduction.

Let $2 = \{0, 1\}$ and $\mathbb{N} = \{0, 1, 2, \ldots\}$. We define the set of bitstreams (streams for short) by

$$2^\omega = \{\sigma \mid \sigma : \mathbb{N} \to 2\}$$

Individual bitstreams will be denoted by $\sigma = (\sigma(0), \sigma(1), \sigma(2), \ldots)$, also written as $(\sigma_0, \sigma_1, \sigma_2, \ldots)$. The following convention will be useful: we include the set 2 into the set 2^ω by putting $[0] = (0, 0, 0, \ldots)$ and $[1] = (1, 0, 0, 0, \ldots)$. For a stream $\sigma \in 2^\omega$, we call $\sigma(0)$ the *initial value* and we define the *(stream) derivative* of σ by $\sigma' = (\sigma(1), \sigma(2), \sigma(3), \ldots)$. (In computer science, initial value and derivative of a stream are better known as *head* and *tail*.)

Initial value and derivative define in fact two functions $(-)(0) : 2^\omega \to 2$ and $(-)' : 2^\omega \to 2^\omega$, which turn the set 2^ω into a *coalgebra* $(2^\omega, (-)(0), (-)')$. This coalgebra is *final* in the set of all (similar such) coalgebras, in the following sense: For every triple (S, o, t) consisting of a set S and functions $o : S \to 2$ and $t : S \to$

S, there exists a unique function $h : S \to 2^\omega$ (called a homomorphism of coalgebras) such that, for all $s \in S$, $o(s) = h(s)(0)$ and $(h(s))' = h(t(s))$. (A proof is easy: take for h the function given by $h(s) = (o(s), o(t(s)), o(t(t(s))), \ldots)$, for all $s \in S$, and show that it is the only possible choice.) The theory of coalgebra will play no explicit role in the present paper (see [JR97, Rut01] for more information on coalgebra). We have mentioned it nonetheless, because it shows that the operations of initial value and derivative are universal (precisely in the sense that finality is called a universal property in category theory). Moreover, it is the basis for the proof and definition principles of coinduction, which we introduce next.

Coinductive proofs will be formulated in terms of the following notion. A relation $R \subseteq 2^\omega \times 2^\omega$ is called a *(stream) bisimulation* if, for all $\langle \sigma, \tau \rangle \in R$,

$$(i)\ \ \sigma(0) = \tau(0), \quad \text{and} \quad (ii)\ \ \langle \sigma', \tau' \rangle \in R$$

The following theorem, called the *coinduction proof principle*, has a straightforward proof (omitted here) but is surprisingly powerful.

Theorem 1 *For all $\sigma, \tau \in 2^\omega$: if there is a bisimulation $R \subseteq 2^\omega \times 2^\omega$ with $\langle \sigma, \tau \rangle \in R$, then $\sigma = \tau$.*

Thus in order to prove the equality of two bitstreams, it suffices to construct a bisimulation that relates them. We shall see only a few examples of proofs by coinduction in this paper; for many more, see [Rut01, Rut03].

Next we show how we can define constant bitstreams and functions on bitstreams, much as in mathematical calculus, by *(stream) differential equations*, which specify their initial value and derivative. For instance, $\gamma' = \gamma$ and $\gamma(0) = 1$ defines the constant stream $\gamma = (1, 1, 1, \ldots)$; and the two equations $f(\sigma, \tau)' = g(\sigma', \tau)$, $f(\sigma, \tau)(0) = \sigma(0)$ and $g(\sigma, \tau)' = f(\sigma, \tau')$ $g(\sigma, \tau)(0) = \tau(0)$, define the function $f(\sigma, \tau) = (\sigma(0), \tau(0), \sigma(1), \tau(1), \ldots)$. The following theorem asserts the existence of a unique solution for a large class of differential equations. It may well be skipped at first reading: the only potential difficulty lies in its generality, and for all the concrete differential equations that will follow in this paper, the reader should have no difficulty in proving by elementary means that they have a unique solution.

Theorem 2 *Let Σ be a set of function symbols. Consider the following system of* (stream) differential equations, *one for every $g \in \Sigma$ with arity $r \in \mathbb{N}$:*

$$g(\sigma_1, \ldots, \sigma_r)' = T(\sigma_1, \ldots, \sigma_r, \sigma_1', \ldots, \sigma_r', \sigma_1(0), \ldots, \sigma_r(0))$$

$$g(\sigma_1, \ldots, \sigma_r)(0) = \phi(\sigma_1(0), \ldots, \sigma_r(0))$$

where: (i) $\sigma_1, \ldots, \sigma_r \in 2^\omega$; (ii) T is an arbitrary composition of function symbols from Σ (possibly including g itself), applied to (a subset of) the arguments that are listed (recall that $[a] = (a, 0, 0, 0, \ldots)$ for $a \in 2$); and (iii) $\phi : 2^r \to 2$ is an arbitrary function (or a constant in case $r = 0$). Then there is a unique family of functions $(g : (2^\omega)^r \to 2^\omega)_{g \in \Sigma}$ satisfying the system of equations above.

Proof: There are many ways to prove this theorem. Here we briefly sketch two of them. The first proof is the coalgebraically purest one, since it is phrased entirely in coalgebraic terms, exploiting the basic fact that 2^ω is a final coalgebra. Let Trm be the set of terms built from the function symbols in Σ and from constants c_σ, one for every bitstream σ. The set Trm can be turned into a coalgebra $(Trm, \langle o, t\rangle : Trm \to 2 \times Trm)$ by defining o and t, first on the constants c_σ as $o(c_\sigma) = \sigma(0)$ and $t(c_\sigma) = c_{\sigma'}$, and then by induction on syntactically more complex terms, following the equations of the theorem. By finality, there exists a unique homomorphism from the coalgebra Trm into the coalgebra 2^ω, which assigns to each *syntactic term* $g(c_{\sigma_1}, \ldots, c_{\sigma_r})$ a stream in 2^ω. This stream is then what we define to be the value of $g(\sigma_1, \ldots, \sigma_r)$, where now g denotes a function on bitstreams (of arity r). For details, we refer the reader to [Rut03], where this construction is carried out for (a subset of) the operators of the calculus of streams of real numbers. For a more general treatment of this type of coinductive definitions, see also [Len99, Bar04].

Alternatively, one may wish to exploit the presence of an obvious ultrametric on 2^ω. It is easy to see that the differential equations from the theorem define contractions (due to their shape which ensures that they are *guarded*). By Banach's fixed point theorem, the solutions of the equations are then obtained as the unique fixed points of these contractions. For examples of this type of use of (ultra)metrics in theoretical computer science, see [AN80, BZ82, SHLG94].

Note that Theorem 2 gives us a very general definition principle, which is of a syntactic nature: it suffices to look at the syntactic format of the differential equations to be sure that they have a unique solution. Theorem 2 is sufficient for our present purposes but can be generalised in many different ways. For instance, the term T above could also be applied to higher-order derivatives of the arguments of g (as in $even(\sigma)' = even(\sigma'')$, $even(\sigma)(0) = \sigma(0)$, which defines $even(\sigma) = (\sigma(0), \sigma(2), \sigma(4), \ldots)$).

3 Boolean algebra

The first algebraic structure that we consider is $(2^\omega, \vee, \wedge, \neg, [0], \langle 1\rangle)$. It has the operations of *or*, *and*, *negation*, and the constants $[0] = (0, 0, 0, \ldots)$ and $\langle 1\rangle = (1, 1, 1, \ldots)$. It inherits its Boolean algebra structure from the Boolean operators on $2 = \{0, 1\}$, which we denote as follows: for all $a, b \in 2 = \{0, 1\}$, $a \vee b = \max\{a, b\}$, $a \wedge b = \min\{a, b\}$, $\neg a = 1 - a$. With these, the Boolean operators on bitstreams can be defined as follows: for all $\sigma, \tau \in 2^\omega$, $n \geq 0$,

$$(\sigma \vee \tau)(n) = \sigma(n) \vee \tau(n)$$

$$(\sigma \wedge \tau)(n) = \sigma(n) \wedge \tau(n)$$

$$(\neg\sigma)(n) = \neg(\sigma(n))$$

The constants are the neutral elements for *or* and *and*: $[0] + \sigma = \sigma$ and $\langle 1\rangle \wedge \sigma = \sigma$, and we have the familiar laws from Boolean algebra such as $\neg(\sigma \vee$

$\tau) = \neg\sigma \wedge \neg\tau$, $\sigma \vee \sigma = \sigma$, and the like. The Boolean operators on bitstreams can, equivalently, be defined as the unique solutions of the following system of differential equations:

derivative:	initial value:	name:
$(\sigma \vee \tau)' = \sigma' \vee \tau'$	$(\sigma \vee \tau)(0) = \sigma_0 \vee \tau_0$	or
$(\sigma \wedge \tau)' = \sigma' \wedge \tau'$	$(\sigma \wedge \tau)(0) = \sigma_0 \wedge \tau_0$	and
$(\neg\sigma)' = \neg(\sigma')$	$(\neg\sigma)(0) = \neg(\sigma_0)$	negation

4 Kleene algebra

Next we consider the structure

$$A_K = (2^\omega, \vee, \cdot, (-)^*, [0], [1])$$

It has constants $[0] = (0, 0, 0, \ldots)$ and $[1] = (1, 0, 0, 0, \ldots)$, where the latter is not to be confused with the constant $\langle 1 \rangle = (1, 1, 1, \ldots)$ that was introduced in Section 3. The operator $\vee$, here also called *sum*, is as before, and we define the *concatenation product* of streams σ and τ by

$$(\sigma \cdot \tau)(n) = (\sigma_0 \wedge \tau_n) \vee (\sigma_1 \wedge \tau_{n-1}) \vee \cdots \vee (\sigma_n \wedge \tau_0)$$

for all $n \geq 0$. Finally, we define the operator of *Kleene star* by

$$\sigma^* \quad = \quad [1] \vee \sigma \vee (\sigma \cdot \sigma) \vee (\sigma \cdot \sigma \cdot \sigma) \vee \cdots$$

where the operation of *infinite sum* is defined, for families of streams $(\tau_i)_{i \in I}$, by $(\bigvee_I \tau_i)(n) = \max\{\tau_i(n) \mid i \in I\}$, for all $n \geq 0$. Equivalently, these operators can also be defined as the unique solutions of the following system of differential equations:

derivative:	initial value:	name:
$(\sigma \vee \tau)' = \sigma' \vee \tau'$	$(\sigma \vee \tau)(0) = \sigma_0 \vee \tau_0$	or (sum)
$(\sigma \cdot \tau)' = (\sigma' \cdot \tau) \vee ([\sigma_0] \cdot \tau')$	$(\sigma \cdot \tau)(0) = \sigma_0 \wedge \tau_0$	concatenation product
$(\sigma^*)' = \sigma' \cdot \sigma^*$	$(\sigma^*)(0) = 1$	Kleene star

(Note that we write $[\sigma_0]$ to denote the stream $(\sigma_0, 0, 0, 0, \ldots)$, which is consistent with our earlier notation of $[0] = (0, 0, 0, \ldots)$ and $[1] = (1, 0, 0, \ldots)$.) Without the star, the above structure is a *semi-ring* [BR88] (which is more or less a ring without subtraction): sum is commutative, sum and product are associative and have neutral elements [0] and [1]: $\sigma \vee [0] = \sigma$, $\sigma \cdot [0] = [0]$, $\sigma \cdot [1] = \sigma$, and product distributes over sum in the usual way. In this semiring, $\vee$ is moreover idempotent and $\cdot$ is commutative: $\sigma \vee \sigma = \sigma$ and $\sigma \cdot \tau = \tau \cdot \sigma$. The presence of star makes of the above structure moreover a *Kleene algebra* [Koz94], satisfying the following laws, the first of which is particularly simple because of the idempotency of sum and the commutativity of product: for all $\sigma, \tau, \rho \in 2^\omega$,

$$\begin{aligned} (\sigma \vee \tau)^* \quad &= \quad \sigma^* \cdot \tau^* \\ \sigma^* \quad &= \quad [1] \vee (\sigma \cdot \sigma^*) \\ \sigma = (\rho \cdot \sigma) \vee \tau \quad &\Rightarrow \quad \sigma = \rho^* \cdot \tau \end{aligned} \tag{1}$$

where the latter law (1) only holds if $\rho(0) = 0$. If the streams ρ and τ are given, the left equality of (1) can be considered as an equation in one unknown, σ. The right equality shows the (unique) solution of this equation, by expressing σ in terms of ρ and τ. (As we shall see later, such equations arise naturally when studying digital circuits, in Section 7.)

Let us next introduce a constant stream that will prove very useful in the definitions of many operators on streams. Also it will be crucial for our semantic description of registers and feedback loops in Boolean circuits, in Section 7. We define:

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

If we define $X^0 = [1]$ and $X^{n+1} = X \cdot X^n$, then $X^2 = (0, 0, 1, 0, 0, 0, \ldots)$, $X^3 = (0, 0, 0, 1, 0, 0, 0, \ldots)$, and so on. Using the operation of infinite sum that was introduced above, we also have:

$$\sigma \quad = \quad [\sigma_0] \vee ([\sigma_1] \cdot X) \vee ([\sigma_2] \cdot X^2) \vee \cdots = \bigvee [\sigma(n)] \cdot X^n$$

This formula shows that streams σ are formal power series in one formal variable (namely, the constant X). Also, it shows that the Kleene algebra of the present section can be understood in formal language theoretical terms. We can associate with a stream σ the formal language $\{X^n \mid \sigma(n) = 1\}$, that is, a set of finite words over the one-letter alphabet X. With this association, the constant stream [0] corresponds to the empty language; the stream [1] corresponds to the language $\{\varepsilon\}$ containing the empty word $X^0 = \varepsilon$; and the operations of sum, product, and star correspond to the familiar operations of formal language theory: union, concatenation, and Kleene star.

Here are a few examples of streams defined by terms in the present Kleene algebra. For any $n \geq 0$ and $a_0, \ldots, a_n \in 2$, we have

$$[a_0] \vee ([a_1] \cdot X) \vee \ldots \vee ([a_n] \cdot X^n) = (a_0, a_1, \ldots, a_n, 0, 0, 0, \ldots)$$

We call such streams *polynomial*. Now if we define

$$\langle a_0 a_1 \cdots a_n \rangle = (\, [a_0] \vee ([a_1] \cdot X) \vee \ldots \vee ([a_n] \cdot X^n)\,) \cdot (X^{n+1})^*$$

then it is easy to show that $\langle a_0 a_1 \cdots a_n \rangle' = \langle a_1 \cdots a_n a_0 \rangle$, whence

$$\langle a_0 a_1 \cdots a_n \rangle = (a_0, a_1, \ldots, a_n, a_0, a_1, \ldots, a_n, \ldots)$$

Such streams we call *periodic*. For instance,

$$\langle 010 \rangle = X \cdot (X^3)^* = (0, 1, 0, 0, 1, 0, 0, 1, 0, \ldots)$$

(Our earlier notation for $\langle 1 \rangle = (1, 1, 1, \ldots)$ is consistent with the present use of sharp angular brackets, since according to the definition above, $\langle 1 \rangle = [1] \cdot X^* = X^* = (1, 1, 1, \ldots)$.) For a last example, we note that the concatenation product of X^* with a stream σ yields

$$X^* \cdot \sigma \quad = \quad (\sigma(0), \sigma(0) \vee \sigma(1), \sigma(0) \vee \sigma(1) \vee \sigma(2), \ldots) \qquad (2)$$

that is, the stream of all partial sums of σ. This stream will appear in the semantics of a feedback circuit in Section 7.

5 Sum and product modulo-2

Let the operation of *addition modulo-2* (aka exclusive or) be defined as usual: $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$, for all $a, b \in 2 = \{0, 1\}$. In this section, we consider the structure

$$A_{mod2} = (2^\omega, \oplus, \ominus, \otimes, \oslash, [0], [1])$$

consisting of the set of bitstreams with (again the constants $[0] = (0, 0, 0, \ldots)$ and $[1] = (1, 0, 0, 0, \ldots)$ and) the operations of sum, minus, product and inverse, all *modulo-2*, which we define, for all $\sigma, \tau \in 2^\omega$, $n \geq 0$, as follows:

$$\begin{aligned} (\sigma \oplus \tau)(n) &= \sigma_n \oplus \tau_n \\ (\ominus\sigma)(n) &= \sigma_n \\ (\sigma \otimes \tau)(n) &= (\sigma_0 \wedge \tau_n) \oplus\ (\sigma_1 \wedge \tau_{n-1}) \oplus \cdots \oplus\ (\sigma_n \wedge \tau_0) \end{aligned}$$

Note that we use the same symbol for the addition modulo-2 of Booleans and streams. Also note that $\ominus\sigma = \sigma$, as $\sigma \oplus \sigma = [0]$. The operator of inverse modulo-2 is defined only for streams σ with $\sigma_0 = 1$, that is, streams of the form $\sigma = [1] \oplus (X \otimes \tau)$, for arbitrary $\tau \in 2^\omega$ (recall that $X = (0, 1, 0, 0, 0, \ldots)$ and note that $X \otimes \tau = (0, \tau_0, \tau_1, \tau_2, \ldots)$). For such streams, we define

$$1 \oslash ([1] \oplus (X \otimes \tau)) \;=\; [1] \oplus (X \otimes \tau) \oplus ((X \otimes \tau) \otimes (X \otimes \tau)) \oplus \cdots$$

where the infinite sum is defined similarly to the infinite sum of Section 4. Equivalently, these operators can be defined as the unique solutions of the following systems of differential equations:

derivative:	initial value:	name:
$(\sigma \oplus \tau)' = \sigma' \oplus \tau'$	$(\sigma \oplus \tau)(0) = \sigma_0 \oplus \tau_0$	sum modulo-2
$(\ominus\sigma)' = \ominus(\sigma')$	$(\ominus\sigma)(0) = \sigma_0$	minus modulo-2
$(\sigma \otimes \tau)' = (\sigma' \otimes \tau) \oplus ([\sigma_0] \otimes \tau')$	$(\sigma \otimes \tau)(0) = \sigma_0 \wedge \tau_0$	product modulo-2
$(1 \oslash \sigma)' = \sigma' \otimes (1 \oslash \sigma)$	$(1 \oslash \sigma)(0) = 1$	inverse mod-2 ($\sigma_0 = 1$)

We shall write $\sigma \oslash \tau$ for $\sigma \otimes (1 \oslash \tau)$. (The notation $\frac{\sigma}{\tau}$ will *not* be used to denote $\sigma \oslash \tau$, but is reserved for the operation of division in the algebra A_{2adic}, in Section 6.)

The above structure is a ring in which sum is idempotent and product is commutative, with [0] and [1] as the neutral elements. The structure is moreover an integral domain (that is, it has no zero-divisors) since $\sigma \neq [0]$ and $\tau \neq [0]$ imply $\sigma \otimes \tau \neq [0]$. Furthermore, the operation of $1 \oslash \sigma$ acts as an inverse to product: $\sigma \otimes (1 \oslash \sigma) = [1]$ for all σ with $\sigma_0 = 1$. For $\oslash$ we have the usual properties (such as $(1 \oslash \sigma) \otimes (1 \oslash \tau) = 1 \oslash (\sigma \otimes \tau)$).

The operators in the present section will be used to describe flip-flops and other circuits, in Section 7.

6 The 2-adic operators

We introduce the algebra

$$A_{2adic} = (2^\omega, +, -, \times, /, [0], [1])$$

consisting of the set of streams and the usual constants, now with the *2-adic operators* of addition, minus, product and inverse (division). The main motivation for these operators lies in their use in binary arithmetic on bitstreams (as we shall see shortly), which is also the reason why in the presence of these operators, the elements of 2^ω are sometimes called the *2-adic numbers* [Kob77, Vui94].

There are various ways of defining the 2-adic operators. A quick and coalgebraically clean way is to take them as the unique solution of the following system of differential equations, which is a variation on the systems that we have seen in the preceding sections:

derivative:	initial value:	name:
$(\sigma + \tau)' = (\sigma' + \tau') + [\sigma_0 \wedge \tau_0]$	$(\sigma + \tau)(0) = \sigma_0 \oplus \tau_0$	2-adic sum
$(-\sigma)' = -(\sigma' + [\sigma_0])$	$(-\sigma)(0) = \sigma_0$	2-adic minus
$(\sigma \times \tau)' = (\sigma' \times \tau) + ([\sigma_0] \times \tau')$	$(\sigma \times \tau)(0) = \sigma_0 \wedge \tau_0$	2-adic product
$(1/\sigma)' = -(\sigma' \times (1/\sigma))$	$(1/\sigma)(0) = 1$	2-adic inverse ($\sigma_0 = 1$)

As usual, we shall write σ/τ or $\frac{\sigma}{\tau}$ for $\sigma \times (1/\tau)$; also we write $-\sigma$ for $-[1] \times \sigma$.

The sum $\sigma + \tau$ is computed by elementwise addition but with the proviso that 'overflow' bits are carried over to higher-order positions (that is to say, next right in the stream). This explains the initial value $(\sigma + \tau)(0) = \sigma_0 \oplus \tau_0$ and the presence of the 'carry' term $[\sigma_0 \wedge \tau_0]$ in the derivative $(\sigma + \tau)'$. (Note that as a consequence, there is no easy formula for the n-th value $(\sigma + \tau)(n)$.) The definition of minus is completely determined by the 'requirement' that $\sigma + (-\sigma) = [0]$: taking initial values on both sides implies that $\sigma_0 \oplus ((-\sigma)(0)) = 0$, whence $(-\sigma)(0) = \sigma_0$; and taking derivatives on both sides implies $\sigma' + (-\sigma)' + [\sigma_0] = [0]$, from which the shape of the differential equation for $-\sigma$ follows. Multiplication is defined in terms of shift and addition. In fact, the above equation expresses the 2-adic product in terms of the 2-adic sum in precisely the same way as the corresponding equation in Section 5 expresses product modulo-2 in terms of addition modulo-2. The definition of inverse, finally, follows from the 'requirement' that $\sigma \times (1/\sigma) = [1]$, for any $\sigma \in 2^\omega$ with $\sigma_0 = 1$, again by taking initial values and derivatives on both sides.

The structure $(2^\omega, +, -, \times, /, [0], [1])$ is again a commutative ring and integral domain. Sum, however, is no longer idempotent, and there are some unusual, surprising facts, such as

$$-[1] \quad = \quad (1, 1, 1, \ldots) \tag{3}$$

(which equals $1/([1] - X)$) and $\sigma + \sigma = X \times \sigma$.

We mentioned above that in the present context, bitstreams are also called 2-adic numbers. This can be explained by viewing (certain) bitstreams as the binary representations of (certain) numbers, as follows. For $n \in \mathbb{N}$, we define the *binary representation* $B(n)$ as the unique stream satisfying the following system of differential equations (one for each n):

$$(B(n))' = B((n - odd(n))/2), \quad (B(n))(0) = odd(n)$$

where $odd(n) = 1$, if n is odd, and $= 0$ if n is even. (Note that we are using the same symbols for the operators on numbers and streams.) This defines the usual binary representation of the natural numbers, with infinitely many 0's appended at the end. For instance, $B(13) = (1, 0, 1, 1, 0, 0, 0, \ldots)$ (note that the least significant bit is on the left). The definition of B can be extended to the set Z of all integers and even to the set $\hat{Q} = \{n/(2m + 1) \mid n, m \in Z\}$ of all rationals with odd denominator, by using the same equations as before: we put $(B(q))' = B((q - odd(q))/2)$ and $(B(q))(0) = odd(q)$, for all $q \in \hat{Q}$, where now $odd(n/2m + 1) = odd(n)$. Viewed as a function $B : \hat{Q} \to 2^\omega$, it is in fact a homomorphism of integral domains, since $B(0) = [0]$, $B(1) = [1]$ and, for all $n, m \in Z$,

$$\begin{aligned} B(n + m) &= B(n) + B(m) \\ B(-n) &= -B(n) \\ B(n \times m) &= B(n) \times B(m) \\ B(n/2m + 1) &= B(n)/B(2m + 1) \end{aligned}$$

In Section 7, the 2-adic operators will be used in circuits for binary arithmetic.

7 Bitstreams and circuits

We show how bitstreams and the various bitstream operators can be used to define the semantics of *Boolean (digital) circuits*, in terms of functions from (input) streams to (output) streams. Circuits without feedback loops and memory (sometimes called *logical* circuits) can be described by means of *Boolean functions* (functions from Booleans to Booleans), which have been broadly studied and are well understood [Weg87]. Here we use bitstreams, that is, infinite sequences of Booleans, instead of just Booleans, because our circuits will in general be *sequential*: they have feedback loops and contain memory (in the form of registers). As a consequence, output values may depend on previous input values.

As we shall see, digital circuits will in general be built from copiers (aka splitters); wires; basic gates for the Boolean connectives *and*, *or*, and *negation*; registers (for memory); and they will contain feedback loops. Their semantics will typically involve operators from two or more of the four bitstream algebras that we have considered sofar. We have no systematic results for this most general case, but we shall consider a number of examples. (In Section 9, we shall look at a special class of so-called *linear* circuits, for which we *do* have a complete characterisation.) The relevance of these examples lies in the fact that they lead to mixed algebraic expressions and thereby motivate the use of *mixed laws*, involving operators from different algebras. We shall use a number of such laws in the present section, and prove them (and several other ones), by coinduction, in Section 8.

Circuits will generally have a number of *input ends*, denoted by arrow shafts $\vdash\!\!\!\relbar\!\!\!\relbar$, and a number of *output ends*, denoted by arrow heads $\longrightarrow$. Often, we shall concentrate on circuits that have exactly one input and one output end, just for convenience and without loss of generality. For streams $\sigma, \tau \in 2^\omega$, we shall write

$$\begin{tikzcd} \sigma \arrow[r, |-] & \cdots \arrow[r] & \tau \end{tikzcd}$$

and say that the circuit *inputs* the stream σ and *outputs* the stream τ. Typically, the output is a function of the input: $\tau = f(\sigma)$, for some $f : 2^\omega \to 2^\omega$, and we say that the circuit *implements* the function f. Operationally, the behaviour of a circuit consists of an infinite sequence of actions, at time moments $0, 1, 2, \ldots$. At each moment $n \geq 0$, the circuit simultaneously (also called *synchronously*) inputs the value $\sigma_n \in 2$ at its input end and outputs the value $\tau_n \in 2$ at its output end. In general, the value τ_n will depend on current and previous values σ_i, for $i \leq n$. Here are the basic gates, out of which all circuits will be constructed:

(a) *Wires* are basic gates with one input end and one output end, and are either fully connected or fully disconnected:

$$\begin{tikzcd} \sigma \arrow[r, |-] & 1 \arrow[r] & \sigma & \sigma \arrow[r, |-] & 0 \arrow[r] & {[0]} \end{tikzcd}$$

(b) A *copier* simply produces two identical copies of its input:

$$\begin{tikzcd} & & \sigma \\ \sigma \arrow[r, |-] & c \arrow[ur, bend left] \arrow[dr, bend right] & \\ & & \sigma \end{tikzcd}$$

(c) An *or* (or *sum*) gate takes two input streams σ and τ and produces their sum as output:

$$\begin{tikzcd} \sigma \arrow[dr, |-, bend left] & & \\ & \vee \arrow[r] & (\sigma \vee \tau) \\ \tau \arrow[ur, |-, bend right] & & \end{tikzcd}$$

(d) An *and* gate takes two input streams σ and τ and produces $\sigma \wedge \tau$ as output:

$$\begin{tikzcd} \sigma \arrow[dr, |-, bend left] & & \\ & \wedge \arrow[r] & (\sigma \wedge \tau) \\ \tau \arrow[ur, |-, bend right] & & \end{tikzcd}$$

(e) An *inverter* or *negation* gate takes an input stream σ and produces its negation as output:

$$\sigma \vdash\!\!\xrightarrow{\;\neg\;} \neg\sigma$$

(f) A *register* gate is defined by

$$\sigma \vdash\!\!\xrightarrow{\;R\;} (0, \sigma_0, \sigma_1, \sigma_2, \ldots)$$

It can be viewed as a one-place memory cell. Initially, it contains 0, which is the first value to be output and after which the elements of σ follow, delayed by one time step. In our algebraic modelling of the register, we have three options, because of the following mixed law:

$$X \cdot \sigma \;=\; X \otimes \sigma = \; X \times \sigma = \; (0, \sigma_0, \sigma_1, \sigma_2, \ldots) \tag{4}$$

As we shall see below, it will depend on the context which operator can be used best.

Note that all of the above basic gates but the register can be modelled within Boolean algebra (and without the need for streams). It is the use of registers that forces us to move from Boolean algebra to (one or more of) the other algebras introduced in the previous sections.

We shall also consider the following two sum gates, which can be considered as 'macro' gates and can be constructed out of the basic gates above (details will be given in Section 9):

$$\begin{tikzcd}
\sigma \arrow[dr, maps to, bend left] & & & \sigma \arrow[dr, maps to, bend left] & & \\
& \oplus \arrow[r] & (\sigma \oplus \tau) & & + \arrow[r] & (\sigma + \tau) \\
\tau \arrow[ur, maps to, bend right] & & & \tau \arrow[ur, maps to, bend right] & &
\end{tikzcd}$$

They take inputs σ and τ and produce outputs $\sigma \oplus \tau$ and $\sigma + \tau$, which are the respective sum operators from the algebras A_{mod2} and A_{2adic}.

As we mentioned before, we shall typically consider *sequential* circuits, allowing feedback. Feedback loops are always required to pass through (at least) one register. As we shall see, this will ensure that the corresponding semantic equations will be well-formed (*guarded*, with a technical term), so that the circuit will have a well-defined behaviour. Here is a first example:

$$\begin{tikzcd}
& \circ \arrow[d, maps to] & & \circ \arrow[ll, maps to, "R"'] & \\
\sigma \arrow[r, maps to] & \vee \arrow[r] & \circ \arrow[r, maps to] & c \arrow[u] \arrow[r] & \tau
\end{tikzcd}$$

It is a variation on a flip-flop circuit that we shall see below. The explicit denotation of composition by the symbol $\circ$ in the circuit above may seem a bit pedantic but will allow us to compute the input/output behaviour of circuits by systematically replacing the $\circ$ symbols by 'internal' streams, as follows:

$$\begin{tikzcd}
& \rho_1 \arrow[d, mapsto, no head] & & \rho_2 \arrow[ll, mapsto, "R" description] & \\
\sigma \arrow[r, mapsto, no head] & \vee \arrow[r] & \rho_3 \arrow[r, mapsto, no head] & c \arrow[u] \arrow[r] & \tau
\end{tikzcd}$$

Each of the basic gates in the circuit above gives rise to a semantic equation:

$$\tau = \rho_3 = \rho_2, \quad \rho_3 = \sigma \vee \rho_1, \quad \rho_1 = X \cdot \rho_2$$

Note that because of the presence of the *or* gate, we have used the concatenation product from Kleene algebra A_K in the equation for the register (instead of the two alternatives mentioned in identity (4)). Next we can express the output τ in terms of the input σ, by solving the above system of equations in A_K, eliminating the 'internal' variables ρ_i. This gives $\tau = (X \cdot \tau) \vee \sigma$. Applying identity (1) from Section 4 gives $\tau = X^* \cdot \sigma$ and, by identity (2) from Section 4,

$$\tau \quad = \quad (\sigma(0), \sigma(0) \vee \sigma(1), \sigma(0) \vee \sigma(1) \vee \sigma(2), \ldots)$$

Thus this circuit outputs 0's until the arrival of the first 1, after which it will output always 1's.

The following flip-flop circuit is an obvious variation on the one above:

$$\begin{tikzcd}
& \rho_1 \arrow[d, mapsto, no head] & & \rho_2 \arrow[ll, mapsto, "R" description] & \\
\sigma \arrow[r, mapsto, no head] & \oplus \arrow[r] & \rho_3 \arrow[r, mapsto, no head] & c \arrow[u] \arrow[r] & \tau
\end{tikzcd}$$

As before, it gives rise to three equations:

$$\tau = \rho_3 = \rho_2, \quad \rho_3 = \sigma \oplus \rho_1, \quad \rho_1 = X \otimes \rho_2$$

but note our choice for $\otimes$ in the equation for the register, which allows us to solve this system within the algebra A_{mod2}. This gives $\tau = (X \otimes \tau) \oplus \sigma$, and $([1] \oplus X) \otimes \tau = \sigma$, recalling that $\ominus\rho = \rho$ for all $\rho \in 2^\omega$. As a consequence, we obtain

$$\tau = \; (1 \oslash ([1] \oplus X)) \otimes \sigma = \; (\sigma(0), \sigma(0) \oplus \sigma(1), \sigma(0) \oplus \sigma(1) \oplus \sigma(2), \ldots)$$

(The latter equality follows easily from the definitions of the operators and the observation that $1 \oslash ([1] \oplus X) = (1, 1, 1, \ldots)$.) We see that this circuit acts as a switch (flip-flop), outputting 1's only when an odd number of 1's has been input.

Using the same feedback circuit once again, but now with the macro gate for 2-adic addition:

$$\begin{tikzcd}
& \rho_1 \arrow[d, mapsto, no head] & & \rho_2 \arrow[ll, mapsto, "R" description] & \\
\sigma \arrow[r, mapsto, no head] & + \arrow[r] & \rho_3 \arrow[r, mapsto, no head] & c \arrow[u] \arrow[r] & \tau
\end{tikzcd}$$

gives rise to the equations

$$\tau = \rho_3 = \rho_2, \ \ \rho_3 = \sigma + \rho_1, \ \ \rho_1 = X \times \rho_2$$

Calculating in the algebra A_{2adic} gives

$$\tau = \ \frac{1}{[1] + X} \times \sigma = \ -\sigma$$

where the latter equality follows from $\frac{1}{[1]+X} = (1, 1, 1, \ldots) = -[1]$. So we see that the circuit above computes for a 2-adic number its 2-adic minus.

The semantics of each of the three example circuits above was given within one and the same algebra. Next we look at some examples that will involve some basic reasoning with mixed terms, containing operators from two or more different algebras. Our first example of this type is again a feedback loop, now without input and using a not-gate:

$$\begin{tikzcd}
& \rho_1 \arrow[dl, "R"'] & & \\
\rho_2 \arrow[dr, "\neg"'] & & c \arrow[ul] \arrow[r] & \tau \\
& \rho_3 \arrow[ur] & &
\end{tikzcd}$$

Note that even though this circuit has no inputs, it produces an output stream τ because of the presence of the register R that contains (by definition) an initial value 0. We obtain the following equations:

$$\tau = \rho_1 = \rho_3, \ \ \rho_3 = \neg\rho_2, \ \ \rho_2 = X \odot \rho_1$$

where $\odot$ in the equation for the register is one of $\{\cdot, \ \otimes, \ \times\}$, since we have again three options for the equation for the register, according to mixed law (4). These three equations together imply

$$\tau \ \ = \ \ \neg(X \odot \tau) \tag{5}$$

As it turns out, we are able to solve this equation in each of our algebras, because of the presence of the following three mixed laws that relate the operator $\neg$ with the respective product operators: for all $\sigma \in 2^\omega$,

$$\begin{aligned}
\neg(X \cdot \sigma) \ &= \ [1] \vee (X \cdot \neg\sigma) && (6)\\
\neg\sigma \ &= \ (1 \oslash ([1] \oplus X)) \oplus \sigma && (7)\\
\neg\sigma \ &= \ -([1] + \sigma) && (8)
\end{aligned}$$

(As we indicated above, these and other mixed laws will be proved only in Section 8.) We can now compute with (5) and each of these laws as follows:

(i) $\tau = \neg(X \cdot \tau)$ and (6) imply

$$\begin{aligned}
\tau \ &= \ [1] + (X \cdot \neg\tau)\\
&= \ [1] + (X \cdot \neg(\neg(X \cdot \tau)))\\
&= \ [1] + (X^2 \cdot \tau)
\end{aligned}$$

Using identity (1), from Section 4, we find $\tau = (X^2)^*$.

(ii) $\tau = \neg(X \otimes \tau)$ and (7) imply

$$\tau = (1 \oslash ([1] \oplus X)) \oplus (X \otimes \tau)$$

As a consequence, $([1] \oplus X) \otimes \tau = 1 \oslash ([1] \oplus X)$ and

$$\begin{aligned} \tau &= (1 \oslash ([1] \oplus X)) \otimes (1 \oslash ([1] \oplus X)) \\ &= 1 \oslash ([1] \oplus X^2) \quad [\text{since } X \oplus X = [0]] \end{aligned}$$

(iii) $\tau = \neg(X \times \tau)$ and (8) imply $\tau = -([1] + (X \times \tau))$. This gives $([1] + X) \times \tau = -[1]$, whence

$$\begin{aligned} \tau &= \frac{-[1]}{[1] + X} \\ &= \frac{1}{([1] + X) \times ([1] - X)} \quad [\text{since } -[1] = \tfrac{1}{[1]-X}] \\ &= \frac{1}{[1] - X^2} \end{aligned}$$

Of course, the three resulting algebraic expressions should be equal, and one can easily verify that they are:

$$\tau = (X^2)^* = 1 \oslash ([1] \oplus X^2) = \frac{1}{[1] - X^2} = (1, 0, 1, 0, 1, 0, \ldots)$$

Here is a last example circuit, this time using an *and* gate:

$$\begin{tikzcd}
 & \circ \arrow[d, mapsto] & \circ \arrow[l, mapsto, "\neg"] & & \circ \arrow[ll, mapsto, "R"] & \circ \arrow[l, mapsto, "\neg"] & \\
\sigma \arrow[r, mapsto] & \wedge \arrow[rr] & & \circ \arrow[rr, mapsto] & & c \arrow[r] \arrow[u] & \tau
\end{tikzcd}$$

We use the following mixed law (which is a variation on law (6) above): for all $\rho \in 2^\omega$,

$$\neg([1] \vee (X \cdot \rho)) \quad = \quad X \cdot \neg\rho \qquad (9)$$

The circuit above produces an output stream τ satisfying $\tau = \sigma \wedge (\neg(X \cdot \neg\tau))$. Taking the negation of both sides gives $\neg\tau = \neg\sigma \vee (X \cdot \neg\tau)$. Thus $\neg\tau = X^* \cdot \neg\sigma$, by (1). Although this still is a mixed expression, containing operators from both Boolean and Kleene algebra, it may be considered a satisfactory answer, because of the following equalities:

$$\begin{aligned} \tau &= \neg(X^* \cdot \neg\sigma) \\ &= \neg(\neg\sigma(0), \neg\sigma(0) \vee \neg\sigma(1), \neg\sigma(0) \vee \neg\sigma(1) \vee \neg\sigma(2), \ldots) \\ &= (\sigma(0), \sigma(0) \wedge \sigma(1), \sigma(0) \wedge \sigma(1) \wedge \sigma(2), \ldots) \end{aligned}$$

The examples presented in this section illustrate how to use the various bitstream algebras to give a precise description of the input/output behaviour of general sequential circuits. They also show the relevance of mixed laws (collected in Section 8). At the same time, there is a clear need for the algebraic treatment of more such examples, leading to more mixed laws and, ideally, to classifications of both.

8 Mixed laws

We present various mixed laws, involving operators from two or more bitstream algebras. Some of these laws were motivated and applied in the semantics of example circuits, in Section 7. Similar such examples can be given for the other laws. All of these mixed laws can be conveniently proved with the coinduction proof principle (introduced in Section 2), by constructing a suitable relation on bitstreams and proving that it is a bisimulation relation. For the latter, we have to check that the relation is closed under the taking of (elementwise) derivatives, which is where we can profit from the use of differential equations in the definitions of the operators.

Theorem 3 *For all $\sigma, \tau \in 2^\omega$, $n \geq 0$,*

$$\begin{array}{rcl}
\neg[0] & = & \langle 1 \rangle = \, -1 \\
\neg[1] & = & X \cdot \langle 1 \rangle \\
X \cdot \sigma & = & X \otimes \sigma = \, X \times \sigma = \, (0, \sigma_0, \sigma_1, \sigma_2, \ldots) \\
\sigma & = & [\sigma_0] \vee (X \cdot \sigma') = \, [\sigma_0] \oplus (X \otimes \sigma') = \, [\sigma_0] + (X \times \sigma') \\
\neg X & = & [1] \vee (X^2 \cdot \langle 1 \rangle) \\
(X^{n+1})^* & = & 1 \oslash ([1] \oplus X^{n+1}) = \, \dfrac{1}{[1] - X^{n+1}} \\
\neg\sigma & = & [\neg\sigma_0] \vee (X \cdot \neg\sigma) = \, (1 \oslash ([1] \oplus X)) \oplus \sigma = \, -([1] + \sigma) \\
-\sigma & = & \sigma \oplus (X \cdot X^* \cdot \sigma) \\
\sigma + \tau & = & (\sigma \vee \tau) + (\sigma \wedge \tau) \\
 & = & (\sigma \oplus \tau) + (X \times (\sigma \wedge \tau))
\end{array}$$

Proof: Some of the above identities are straightforward, others require a little bit of work. All can be easily proved by coinduction. We mention one example: in order to prove $\neg\sigma = -([1] + \sigma)$, consider the following relation $R \subseteq 2^\omega \times 2^\omega$:

$$R = \{\langle \neg\sigma, \, -([1] + \sigma) \rangle \mid \sigma \in 2^\omega\}$$

According to Theorem 1, it is sufficient to show that R is a bisimulation relation. For $\sigma \in 2^\omega$, we have, firstly,

$$\begin{array}{rcl}
(\neg\sigma)(0) & = & \neg\sigma_0 \\
 & = & 1 \oplus \sigma_0 \\
 & = & (-([1] + \sigma))(0)
\end{array}$$

Computing derivatives gives, secondly, $(\neg\sigma)' = \neg(\sigma')$ and

$$\begin{array}{rcl}
(-([1] + \sigma))' & = & -(([1] + \sigma)' + [(-([1] + \sigma))(0)]) \\
 & & \quad [\text{recall that } (-\tau)' = -(\tau' + [\tau_0]), \text{ for all } \tau \in 2^\omega] \\
 & = & -((([0] + \sigma') + [\sigma_0]) + [\neg\sigma_0])
\end{array}$$

$$\begin{aligned} & \quad [\text{since } (\tau+\rho)' = (\tau'+\rho') + [\tau_0 \wedge \rho_0] \text{ and } [1]' = [0]] \\ = & \quad -(\sigma' + [1]) \quad [\text{using } [a] + [\neg a] = [1], \text{ for } a \in 2] \\ = & \quad -([1] + \sigma') \end{aligned}$$

This proves that $\langle (\neg\sigma)', \, (-([1]+\sigma))' \rangle \in R$, and concludes the proof that R is a bisimulation.

9 Linear circuits and rational streams

We introduce the class of 2-adic linear circuits, which are built using the 'macro' +-gate that was introduced, but not formally defined, in Section 7. Below we include the details of its definition. Then we give a complete characterisation of these circuits in terms of the notion of 2-adic rationality.

We call a circuit *2-adic linear* (or *linear* for short) if it is built from the following basic gates:

(a) Fully connected or disconnected wires, copiers, and registers, as before.

(b) The 2-adic sum gate (aka *sequential binary adder*) takes two inputs σ and τ and produces their 2-adic sum $\sigma + \tau$ as output. It is usually constructed out of so-called *half-adder* circuits

$$\begin{tikzcd}
\sigma \arrow[dr, no head] & & \tau \arrow[dl, no head] \\
 & H \arrow[dl] \arrow[r] & \sigma \wedge \tau \\
\sigma \oplus \tau & &
\end{tikzcd}$$

(whose definition is straightforward and omitted) as follows:

$$\begin{tikzcd}
 & & & & \sigma \arrow[dr, no head] & & \tau \arrow[dl, no head] & & & \\
\sigma \arrow[dr, no head] & & & & & H \arrow[dl] \arrow[rrr] & & & \circ \arrow[dr, no head] & \\
 & + \arrow[r] & \sigma + \tau & = & \circ \arrow[dr, no head] & & \circ \arrow[dl, no head] & R \arrow[l] & \circ \arrow[l, no head] & \vee \arrow[l] \\
\tau \arrow[ur, no head] & & & & & H \arrow[dl] \arrow[rrr] & & & \circ \arrow[ur, no head] & \\
 & & & & \sigma + \tau & & & & &
\end{tikzcd}$$

(In our algebraic framework, one can prove, formally and without too much effort, that the circuit on the right indeed produces output $\sigma + \tau$ on input σ and τ.)

(c) A *minus* gate produces on input σ its 2-adic negative $-\sigma$, and can be defined as follows:

$$\begin{tikzcd}
\sigma \arrow[r, no head] & -1 \arrow[r] & -\sigma & = & & \circ \arrow[d, no head] & R \arrow[l] & \circ \arrow[l, no head] & \\
 & & & & \sigma \arrow[r, no head] & + \arrow[r] & \circ \arrow[r, no head] & c \arrow[r] \arrow[u] & -\sigma
\end{tikzcd}$$

(The fact that the circuit on the right produces $-\sigma$ on input σ was proved in Section 7.)

There is some redundancy in this definition of 2-adic linearity. Clearly, the minus gate is not a primitive, since it is defined in terms of the other basic gates. The same applies also to registers, which can be built out of a copier and a $+$-gate as follows:

$$\begin{tikzcd}
 & & \circ \arrow[dr, mapsto] & & \\
\sigma \arrow[r, mapsto] & c \arrow[ur, bend left] \arrow[dr, bend right] & & + \arrow[r] & \sigma + \sigma = X \times \sigma \\
 & & \circ \arrow[ur, mapsto] & &
\end{tikzcd}$$

So all what is needed, really, are copiers, wires, and $+$-gates.

Here is a simple example of a 2-adic linear circuit:

$$\begin{tikzcd}
{} \arrow[r, mapsto, "1"] & \circ \arrow[r, mapsto, "R"] & \circ \arrow[l, bend right=60, "1"'] \arrow[r, mapsto, "R"] & \circ \arrow[ll, bend left=40, "1"] \arrow[r, mapsto, "1"] & {}
\end{tikzcd}$$

In this picture, we have omitted the explicit representation of copier and sum gates, under the following convention: if a (composition) node in the picture of a circuit has both incoming and outgoing arrows, then first the incoming streams have to be added; then the resulting sum is copied and output along each of the outgoing arrows. As before, we consider intermediate streams ρ_1, ρ_2, ρ_3 for each of the connection points:

$$\begin{tikzcd}
\sigma \arrow[r, mapsto, "1"] & \rho_1 \arrow[r, mapsto, "R"] & \rho_2 \arrow[l, bend right=60, "1"'] \arrow[r, mapsto, "R"] & \rho_3 \arrow[ll, bend left=40, "1"] \arrow[r, mapsto, "1"] & \tau
\end{tikzcd}$$

and obtain the following equations:

$$\begin{aligned}
\rho_1 &= \sigma + \rho_2 + \rho_3 \\
\rho_2 &= X \times \rho_1 \\
\rho_3 &= X \times \rho_2 \\
\tau &= \rho_3
\end{aligned}$$

Eliminating the internal streams yields the following solution:

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

Thus the circuit above transforms an input stream by multiplying it with a (for this circuit) fixed stream.

The same happens with *any* (finite) 2-adic linear circuit, as Theorem 4 below asserts. It uses the notion of rationality, which we define next. We shall simply write 0 for $[0] = (0, 0, 0, \ldots)$, 1 for $[1] = (1, 0, 0, 0, \ldots)$, and -1 for

$-[1] = (1,1,1,\ldots)$. Also we shall write, accordingly, $\sigma - \tau$ for $\sigma + (-\tau)$. Now we call a stream π *polynomial* if it is of the form

$$\pi = c_0 + (c_1 \times X) + (c_2 \times X^2) + \cdots + (c_k \times X^k)$$

where all c_i are either 0, 1, or -1. We call a stream σ *2-adic rational* if it is of the form

$$\sigma = \frac{\pi}{\rho}$$

for polynomial streams π and ρ with $\rho(0) = 1$. For instance, the stream $\frac{X^2}{1-X-X^2}$ that we encountered above, is rational. It satisfies

$$\frac{X^2}{1 - X - X^2} = \langle 0011 \rangle = (0,0,1,1,0,0,1,1,0,0,1,1,\ldots)$$

One can easily show that any 2-adic rational stream is ultimately (that is, after a finite prefix) periodic, a fact that is well-known in 2-adic number theory [Kob77, Vui94]. It does not play a role in what follows.

Theorem 4 below is formulated for circuits with precisely one input and one output end, but it can easily be generalised for circuits with many input and output ends. Its shows that any 2-adic linear finite circuit multiplies the 2-adic number that it inputs with a fixed (rational) 2-adic number. Conversely, any such function can be implemented by a finite linear circuit. Although the proof of this theorem constructs circuits that are not necessarily minimal, they are generally very small. Therefore, the theorem can be used to reduce circuits by describing their behaviour by an algebraic expression, from the simplified result of which a smaller circuit can be obtained.

Theorem 4

(a) Every finite 2-adic linear circuit, possibly with feedback loops (which pass through at least one register) implements a stream function $f : 2^\omega \to 2^\omega$ of the form, for all $\sigma \in 2^\omega$: $f(\sigma) = \rho \times \sigma$, for a fixed 2-adic rational stream ρ.

(b) Conversely, any function of this form can be implemented by a finite 2-adic linear circuit.

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 σ and with the output end of C a stream τ. With the output end of each register R_i, we associate a stream α_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 all feedback loops pass through at least one register, there are only finitely many of such paths. This leads to an equation of the form

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

for some $a_i, a_i^j \in \{0, 1, -1\}$. 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 ρ_i. Finally, we play the same game for τ, at the output end of C, as we did for each of the registers. This will yield the following type of expression for τ:

$$\begin{aligned} \tau &= (b_1 \times \alpha_1) + \cdots + (b_k \times \alpha_k) + (b \times \sigma) \\ &= ((b_1 \times \rho_1) + \cdots + (b_k \times \rho_k) + b) \ \times \ \sigma \end{aligned}$$

for some $b, b_i \in \{0, 1, -1\}$, which proves (a). For (b), we consider a function $f(\sigma) = \rho \times \sigma$ with

$$\rho = \frac{r_0 + (r_1 \times X) + (r_2 \times X^2) + (r_3 \times X^3)}{1 + (s_1 \times X) + (s_2 \times X^2) + (s_3 \times X^3)}$$

The general case can be treated similarly. We claim that the following circuit implements the function f:

$$\begin{tikzcd}
\sigma \arrow[r] & \rho_0 \arrow[r, "R"] \arrow[rrrr, bend left, "r_0"] & \rho_1 \arrow[r, "R"] \arrow[l, bend left, "-s_1"] \arrow[rrr, bend left, "r_1"] & \rho_2 \arrow[r, "R"] \arrow[ll, bend left, "-s_2"] \arrow[rr, bend left, "r_2"] & \rho_3 \arrow[r, "r_3"] \arrow[lll, bend left, "-s_3"] & \tau
\end{tikzcd}$$

where we have denoted the output stream by τ and the intermediate streams by $\rho_0, \rho_1, \rho_2, \rho_3$. They satisfy the following equations:

$$\begin{aligned} \rho_0 &= \sigma - (s_1 \times \rho_1) - (s_2 \times \rho_2) - (s_3 \times \rho_3) \\ \rho_1 &= X \times \rho_0, \ \ \rho_2 = X \times \rho_1, \ \ \rho_3 = X \times \rho_2 \\ \tau &= (r_0 \times \rho_0) + (r_1 \times \rho_1) + (r_2 \times \rho_2) + (r_3 \times \rho_3) \end{aligned}$$

It follows that $\tau = \rho \times \sigma$, with ρ as above.

The above result is to the best of our knowledge new. A similar result on mod-2 linear circuits and mod-2 rationality exists in various different forms in the literature, although often presented in a semi-formal symbolic form (cf. [Koh78, Chap. 15]).

References

[AN80] A. Arnold and M. Nivat. Metric interpretations of infinite trees and semantics of nondeterministic recursive programs. *Theoretical Computer Science*, 11(2):181–205, 1980.

[Bar04] F. Bartels. *On generalised coinduction and probabilistic specification formats.* PhD thesis, Vrije Universiteit, Amsterdam, 2004.

[BR88] J. Berstel and C. Reutenauer. *Rational series and their languages*, volume 12 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.

[Brz64] J.A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.

[BZ82] J.W. de Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54(1/2):70–120, 1982.

[GK97] M. Goresky and A. Klapper. Feedback shift registers, combiners with memory, and 2-adic span. *Journal of Cryptology*, 10:111–147, 1997.

[Gol82] S.W. Golomb. *Shift register sequences*. Aegean Park Press, 1982.

[Gou93] F.Q. Gouvêa. *p-adic Numbers*. Springer-Verlag, 1993.

[JR97] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the EATCS*, 62:222–259, 1997. URL: www.cwi.nl/~janr.

[Kob77] N. Koblitz. *p-adic Numbers, p-adic Analysis, and Zeta-Functions*, volume 58 of *Graduate Texts in Mathematics*. Springer-Verlag, 1977.

[Koh78] Z. Kohavi. *Switching and finite automata theory*. McGraw-Hill, 1978.

[Koz94] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110:366–390, 1994.

[Len99] M. Lenisa. From set-theoretic coinduction to coalgebraic coinduction. In B. Jacobs and J.J.M.M. Rutten, editors, *Proceedings of CMCS'99*, volume 19 of *ENTCS*. Elsevier, 1999.

[Rut01] 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 *ENTCS*, pages 1–66. Elsevier Science Publishers, 2001.

[Rut03] 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.

[SHLG94] V. Stoltenberg-Hansen, I. Lindstrom, and E.R. Griffor. *Mathematical Theory of Domains*, volume 22 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1994.

[Vui94] J. Vuillemin. On circuits and numbers. *IEEE Transactions on Computers*, 43(8):868–879, 1994.

[Vui00] J. Vuillemin. Finite circuits are characterized by 2-algebraic truth tables. In *Proceedings of Asian 2000*, volume 1961 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 2000.

[Vui03] J. Vuillemin. Digital algebra and circuits. In *Proceedings of the International Symposium on Verification (Theory and Practice)*, volume 2772 of *Lecture Notes in Computer Science*, pages 100–120. Springer-Verlag, 2003.

[Weg87] I. Wegener. *The complexity of Boolean functions.* Wiley, 1987.