



HAL
open science

A Survey of Compositional Signal Flow Theory

Filippo Bonchi, Pawel Sobociński, Fabio Zanasi

► **To cite this version:**

Filippo Bonchi, Pawel Sobociński, Fabio Zanasi. A Survey of Compositional Signal Flow Theory. *Advancing Research in Information and Communication Technology, AICT-600*, pp.29-56, 2021, 10.1007/978-3-030-81701-5_2. hal-03325995

HAL Id: hal-03325995

<https://inria.hal.science/hal-03325995>

Submitted on 25 Aug 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

A Survey of Compositional Signal Flow Theory [★]

Filippo Bonchi¹[0000-0002-3433-723X], Paweł Sobociński³[0000-0002-7992-9685],
and Fabio Zanasi²[0000-0001-6457-1345]

¹ Università di Pisa, Italy filbonchi@di.unipi.it

² University College London, UK f.zanasi@ucl.ac.uk

³ Tallinn University of Technology, Estonia pawel@cs.ioc.ee

Abstract. Signal flow graphs are combinatorial models for linear dynamical systems, playing a foundational role in control theory and engineering. In this survey, we overview a series of works [15, 3, 18, 16, 31, 17, 10, 11, 13, 63, 51] that develop a compositional theory of these structures, and explore several striking insights emerging from this approach. In particular, the use of *string diagrams*, a categorical syntax for graphical models, allows to switch from the traditional combinatorial treatment of signal flow graphs to an algebraic characterisation. Within this framework, signal flow graphs may then be treated as a fully-fledged (visual) programming language, and equipped with important meta-theoretical properties, such as a complete axiomatisation and a full abstraction theorem. Moreover, the abstract viewpoint offered by string diagrams reveals that the same algebraic structures modelling linear dynamical systems may also be used to interpret diverse kinds of models, such as electrical circuits and Petri nets.

In this respect, our work is a contribution to *compositional network theory* (see e.g. [1, 59, 20, 23, 21, 29, 49, 28, 2, 5, 6, 32, 9, 30, 4, 24, 26, 37, 12, ?]), an emerging multidisciplinary research programme aiming at a uniform compositional study of different sorts of computational models.

Keywords: Signal Flow Graphs · Compositional Semantics · Category Theory · String Diagrams.

1 String Diagrams as Resource Sensitive Syntax

Traditional syntax is often identified with trees. Terms are generated from a *signature* Σ , which contains *generators* (aka *operations*) $\sigma \in \Sigma$, each with an *arity* $ar(\sigma) \in \mathbb{N}$. The arity tells us the number of arguments σ requires. In computer science, syntax is often introduced in as a BNF specification. So, e.g.:

$$t ::= Var \mid \sigma(t_1, t_2, \dots, t_n) \quad (\sigma \in \Sigma \wedge ar(\sigma) = n) \quad (1)$$

[★] The second author was supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001) and the Estonian Research Council grant PRG1210.

where Var is a collection of variables. Derivations in (1) are in 1-1 correspondence with those trees where the internal nodes are labelled with $\sigma \in \Sigma$ with $ar(\sigma)$ children, and the leaves are variables and *constants* (arity 0 generators).

The BNF description makes the recursive nature of syntax explicit. The corresponding proof principle is *structural induction* which is extremely useful in programming language theory. Mathematically, these principles stem from the fact that syntax is *free*: it is the initial algebra of a functor $F_{\Sigma, Var} : \mathbf{Set} \rightarrow \mathbf{Set}$, or, following Lawvere [43], the *free category \mathcal{L}_{Σ} with finite products* on Σ .

The latter description is especially illuminating because it emphasises the central role of finite products, which betrays an underlying assumption about the *cartesianity* of whatever the syntax is meant to express. In particular, variables in scope can be used several times, or not used at all. Thus, the data we operate on is *classical*: it can be copied and discarded at will. Perhaps less obvious is the post-hoc justification of the definition of signature: all operations denote *functions* that have precisely one output. Any operation with two outputs can simply be decomposed into two single-output operations by discarding appropriate outputs. Thus being more liberal with the definition of signature by allowing coarities other than 1 would not increase expressivity.

What is an appropriate notion of syntax when the underlying data is not classical? Or if we want to denote non-functional entities (e.g. relations)? A syntax that is more expressive, in this sense, but retains its recursive specification and the associated principle of structural induction?

The answer is to replace free categories with products with *free props* [44, 41].

Definition 1 (Prop). *A prop (product and permutation category) is a symmetric strict monoidal category with set of objects \mathbb{N} , and the monoidal product on objects is addition: $m \oplus n := m + n$.*

We now give a more concrete description of how terms are constructed. Instead of trees, terms are *string diagrams* [57]: terms built from the generators of Σ , quotiented by topologically intuitive deformations. First, because we are in a resource sensitive setting, we need to be more liberal with our notion of signature.

Definition 2. *A monoidal signature Σ is a collection of generators $\sigma \in \Sigma$, each with an arity $ar(\sigma) \in \mathbb{N}$ and coarity $coar(\sigma) \in \mathbb{N}$.*

Concrete terms can still be given a BNF description, as follows:

$$c ::= \sigma \in \Sigma \quad | \quad (2)$$

$$\square \quad | \quad \text{---} \quad | \quad \text{X} \quad | \quad c \oplus c \quad | \quad c; c \quad (3)$$

Differently from (1), arities and coarities are not handled in the BNF but with the additional structure of (3) and an associated sorting discipline, shown below. We only consider terms that have a sort, which is unique if it exists.

$$\frac{}{\sigma : (ar(\sigma), coar(\sigma))} \quad \frac{}{\square : (0, 0)} \quad \frac{}{\text{---} : (1, 1)} \quad \frac{}{\text{X} : (2, 2)} \quad \frac{c : (n, z) \quad d : (z, m)}{c; d : (n, m)} \quad \frac{c : (n, m) \quad d : (r, z)}{c \oplus d : (n+r, m+z)}$$

The diagrammatic convention for $\sigma \in \Sigma$ is to draw it as a box with $ar(\sigma)$ “dangling wires” on the left and $coar(\sigma)$ on the right:

$$ar(\sigma) \left\{ \begin{array}{|c|} \hline \sigma \\ \hline \end{array} \right\} coar(\sigma).$$

The intuition for the two operations in (3) is given by their diagrammatic conventions:

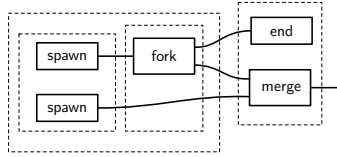
$c; c'$ is drawn $\begin{array}{|c|} \hline c \\ \hline \end{array} \begin{array}{|c|} \hline c' \\ \hline \end{array}$ and $c \oplus c'$ is drawn $\begin{array}{|c|} \hline c \\ \hline c' \\ \hline \end{array}$. The sorting discipline thus keeps track of the “dangling wires” of each term and ensures that the above diagrammatic convention makes sense.

The sorting discipline thus keeps track of the “dangling wires” of each term and ensures that the above diagrammatic convention makes sense.

Example 1. Consider a signature for a simple thread coordination language:

$$\Sigma = \left\{ \begin{array}{|c|} \hline \text{spawn} \\ \hline \end{array}, \begin{array}{|c|} \hline \text{fork} \\ \hline \end{array}, \begin{array}{|c|} \hline \text{merge} \\ \hline \end{array}, \begin{array}{|c|} \hline \text{end} \\ \hline \end{array} \right\}$$

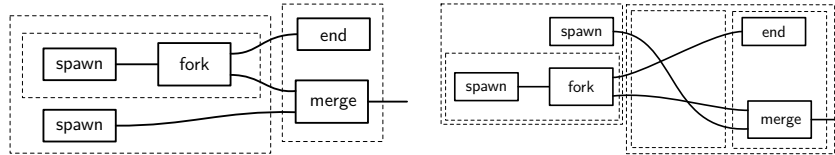
The following is derivable from (2), (3), drawn according to the conventions.



Note that we need to introduce the dotted line boxes in order to identify precisely the term drawn. Indeed the above corresponds to the term

$$((\text{spawn} \oplus \text{spawn}); (\text{fork} \oplus \text{---})); (\text{end} \oplus \text{merge})$$

Note that other terms yield the same connectivity, for example:



which correspond, respectively, to

$$(\text{spawn}; \text{fork}) \oplus \text{spawn}; (\text{end} \oplus \text{merge}) \text{ and}$$

$$(\text{spawn} \oplus (\text{spawn}; \text{fork})) \oplus (((\text{---} \oplus \text{---}); (\text{---} \oplus \text{---})); (\text{end} \oplus \text{merge})).$$

The laws of props identify all and precisely the terms that have the same underlying connectivity. The slogan is “only the topology matters”. This means that, in the free prop on Σ , we do not need to blemish our diagrams with dotted-line boxes, and string diagrams unambiguously represent terms (up-to the laws of props). Nevertheless, we have access to the low-level representation given by (2), (3), with the particular choice of representative immaterial.

Moreover, the fact that our diagrams are the arrows of free props means that they behave analogously to classical syntax. Indeed, as we have seen, they have a recursive definition and enjoy an analogous principle of structural induction.

2 The Calculus of Signal Flow Diagrams

The subject of this paper is a specific string diagrammatic syntax, called the *calculus of signal flow diagrams*. We shall see that, as the name suggests, this syntax is adapted to reason algebraically about *signal flow graphs*, a well-known foundational structure in control theory and engineering (see *e.g.* [58, 45]).

Fixed a semiring \mathbb{R} , the monoidal signature of the calculus of signal flow diagrams is:

$$\Sigma = \{ \bullet, \text{---}\curvearrowright, \text{---}\boxed{x}\text{---}, \text{---}\cup\text{---}, \circ\text{---}, \bullet\text{---}, \text{---}\cup\text{---}, \text{---}\boxed{x}\text{---}, \text{---}\curvearrowleft, \text{---}\circ\text{---} \} \cup \{ \text{---}\boxed{r}\text{---}, \text{---}\boxed{r}\text{---} \mid r \in \mathbb{R} \} \quad (4)$$

The sorts are apparent from the depictions and given by counting dangling wires on each side. As we shall see, the indeterminate x plays a role akin to that in polynomials. Let Circ be the free prop on (4), as in §1. We usually refer to the arrows of Circ (i.e. terms obtained from composing generators in (4)) as *circuits*.

While we delay the semantics of the terms of Circ , to Sections 3 and 4, it is useful to introduce some intuitions for our generators. The five leftmost generators and $\text{---}\boxed{r}\text{---}$ of (4) can be thought of as taking signals—values in \mathbb{R} —from the left boundary to the right: thus \bullet is a *copier*, duplicating the signal arriving on the left; $\text{---}\bullet$ accepts any signal on the left and discards it, producing nothing on the right; $\text{---}\cup\text{---}$ is an *adder* that takes two signals on the left and emits their sum on the right, and $\circ\text{---}$ constantly emits the signal 0 on the right; $\text{---}\boxed{r}\text{---}$ is an *amplifier*, multiplying the signal on the left by $r \in \mathbb{R}$. Finally, $\text{---}\boxed{x}\text{---}$ is a *delay*, a synchronous one place buffer initialised with 0. The five rightmost generators of (4) and $\text{---}\boxed{r}\text{---}$ are “reflected about the y -axis”. Their behaviour is symmetric—here it is helpful to think of signals as flowing from right to left.

Example 2. Consider the two circuits below.



According to our intuitions, in the left circuit, the signal flows from right to left, while in the other from left to right – indeed, the “cup” \bullet ; $\text{---}\cup\text{---}$ and “cap” $\text{---}\cup\text{---}$; $\text{---}\bullet$ allow us to form a feedback loop. In Section 3, we shall provide circuits with a formal semantics. In Example 4 we shall see that the circuits have the same semantics, despite the apparent incompatibility in signal flow direction.

We identify two sub-categories of Circ : $\text{Circ}^{\rightarrow}$ is the free prop on

$$\{ \bullet, \text{---}\curvearrowright, \text{---}\boxed{x}\text{---}, \text{---}\cup\text{---}, \circ\text{---} \} \cup \{ \text{---}\boxed{r}\text{---} \mid r \in \mathbb{R} \}$$

and Circ^{\leftarrow} is the free prop on

$$\{ \bullet, \text{---}\cup\text{---}, \text{---}\boxed{x}\text{---}, \text{---}\curvearrowleft, \text{---}\circ\text{---} \} \cup \{ \text{---}\boxed{r}\text{---} \mid r \in \mathbb{R} \}$$

The notation recalls the intuition that for circuits in $\text{Circ}^{\rightarrow}$, the signal flows from left to right, and in Circ^{\leftarrow} from right to left. Clearly Circ^{\leftarrow} is isomorphic to the opposite category of $\text{Circ}^{\rightarrow}$: Circ^{\leftarrow} circuits can be seen as reflections of $\text{Circ}^{\rightarrow}$ circuits.

2.1 Feedback and Classical Signal Flow Diagrams

Beyond $\text{Circ}^{\rightarrow}$ and Circ^{\leftarrow} , we identify a class of circuits that adheres closely to the orthodox notion of signal flow graph [45], albeit without directed wires. Here, the signal can flow from left to right, as in $\text{Circ}^{\rightarrow}$, but with the possibility of *feedbacks*, provided that these pass through at least one delay. This amounts to defining, for all n, m , a map $\text{Tr}(\cdot): \text{Circ}[n+1, m+1] \rightarrow \text{Circ}[n, m]$ taking $c: n+1 \rightarrow m+1$ to the n -to- m circuit below:



Intuitively, $\text{Tr}(\cdot)$ adds to c a feedback loop from its topmost right to its topmost left port. We can now formally define classical signal flow graphs as the (symmetric monoidal) sub-category SF of Circ inductively given as follows:

- (i) if $c \in \text{Circ}^{\rightarrow}[n, m]$, then $c \in \text{SF}[n, m]$
- (ii) if $c \in \text{SF}[n+1, m+1]$, then $\text{Tr}(c) \in \text{SF}[n, m]$
- (iii) if $c_1 \in \text{SF}[n, z]$ and $c_2 \in \text{SF}[z, m]$, then $c_1 ; c_2 \in \text{SF}[n, m]$
- (iv) if $c_1 \in \text{SF}[n, m]$ and $c_2 \in \text{SF}[r, z]$, then $c_1 \oplus c_2 \in \text{SF}[n+r, m+z]$.

Equivalently, SF is the smallest sub-category of Circ that contains $\text{Circ}^{\rightarrow}$ and is closed under Tr . For instance, the right circuit of Example 2 is in SF .

Remark 1. The reader may have recognised in (5) the structure of a *trace* [38]. In fact, there is a tradition of modelling systems with feedback in terms of traced monoidal categories [34]. The algebraic core of these structures has also been studied in the context of iteration theories [8] and Stefanescu’s network algebra [60]. Concerning the more specific setting of signal flow graphs, previous categorical approaches have been mostly based on coalgebra theory [55, 7, 46]. A distinguishing feature of the string diagrammatic approach presented here (originated in [15], and then independently proposed in [3]) is the adoption of a more abstract framework, Circ , featuring more circuit diagrams than just classical signal flow graphs (which form a subcategory SF). As we will see in Sections 4 and 5, the increased generality comes at the price of making the operational analysis subtler. The payoff is a more elementary syntax (4), not featuring any primitive for recursion, and the possibility of a neat axiomatisation (Section 3.1) in terms of well-known algebraic structures such as bimonoids and Frobenius monoids.

3 Denotational Semantics

Here we equip circuits with a *denotational semantics*, assigning to each an element of an appropriate mathematical universe.

Since each generator of (4) has a clear flow directionality, it may be tempting to interpret them as *functions*, mapping inputs to outputs. However, when “incompatible” connectors are combined, functional interpretations often become

untenable. For instance, there is no clear way of understanding $\bullet - ; - \curvearrowright$ as a function. It can, however, be interpreted as a *relation* that constrains the observable behaviours: for example, in this particular case, the constraint is that at any point in time the *same* value is observed on both ports.

The semantics of a circuit $c \in \text{Circ}[n, m]$ is thus some relation R . A pair $(u, v) \in R$ is *evidence of a possible execution*. The specific nature of u and v changes according to the semantic model chosen, but the idea is that in the execution evidenced by (u, v) , u and v are vectors of observations, with u accounting for what is observed on the left ports and v for what is observed on the right ports. Such relations naturally organise themselves as arrows of a prop.

Definition 3 (Rel_X). *Given a set X of observations, Rel_X is the prop with arrows $m \rightarrow n$ the relations $R \subseteq X^m \times X^n$. The monoidal product is given by cartesian product, and composition of $R : m_1 \rightarrow m_2$ and $S : m_2 \rightarrow m_3$ is:*

$$\{(u, v) \mid u \in R^{m_1}, v \in R^{m_3} \text{ and there is } w \in R^{m_2} \text{ with } (u, w) \in R \text{ and } (w, v) \in S\}$$

Since the syntax (4) is parametrised over a semiring R , and the interpretation of the white generators $\{\curvearrowright, \circ, \curvearrowleft, \circ\}$ is addition, it is natural to consider observations at each port as elements of R , which provides the bare-bone algebraic structure (addition and multiplication) of signals. As alluded to previously, different choices of R allow us to change the semantic model of a circuit. For the remainder of this section we assume that the semiring of signals R is a field k .

The denotational semantics assigns to a circuit c , the relation of *trajectories* that correspond to all possible executions of c . There are, again, different choices for what one may consider as an “execution”, giving us one additional dimension of semantic models. The two cases mainly studied in the literature are: *finite-in-the-past*, *infinite-in-the-future* executions—equivalent to assuming executions start with 0-initialised registers—and *bi-infinite* trajectories, equivalent to (additionally) considering executions where registers can hold arbitrary initial values at all times. We will make these observations more precise in Remark 5, having introduced formal operational semantics of circuits.

The set of *finite-in-the-past*, *infinite-in-the-future* trajectories is the field $k((x))$ of *Laurent series* over k . Indeed, a Laurent series σ can be understood as a *stream*, i.e. an infinite sequence of k -elements

$$\sigma_{-2} \ \sigma_{-1} \ \underline{\sigma_0} \ \sigma_1 \ \sigma_2 \ \dots \ \sigma_n \ \dots$$

where σ_i intuitively represents a discrete unit of signal appearing at one of the circuit ports at a certain moment in time. We underline the *initial* moment of time: thus, in the above, the behaviour σ_0 is observed at time 0. Thus a Laurent series may have possibly infinitely many non-zero observations in the future (σ_1 , σ_2 , etc.), but only finitely many in the past (σ_{-1} and σ_{-2}). An equivalent way of seeing Laurent series is as formal sums $\sigma = \sum_{i=d}^{\infty} \sigma_i x^i$, where $d \in \mathbb{Z}$ indexes the first non-zero element appearing in σ . In this way, Laurent series can be manipulated (added, multiplied) as polynomials. In fact, Laurent series are well-known to be the field of fractions of the ring $k[[x]]$ of formal power series.

The set of *bi-infinite* trajectories is $k^{\mathbb{Z}}$, i.e. the set of functions from the set of integers \mathbb{Z} to k . The algebraic structure common to both $k((x))$ and $k^{\mathbb{Z}}$ is that they are $k[x, x^{-1}]$ modules: roughly speaking, this amounts to saying that:

- trajectories are time-independent: they can be moved one place forward in time (captured as the action of multiplying by x) and moved one place back in time (the action of multiplying by x^{-1});
- trajectories are linear: the always 0 sequence is a trajectory and trajectories are closed under pointwise addition.

We can define the denotational semantics in a trajectory-agnostic way:

Definition 4. Let $\text{Traj} \in \{k((x)), k^{\mathbb{Z}}\}$. The prop morphism $\llbracket \cdot \rrbracket : \text{Circ} \rightarrow \text{Rel}_{\text{Traj}}$ is inductively defined on circuits as follows. For the generators in (4)

$$\begin{array}{ll} \bullet \text{---} \curvearrowright \mapsto \left\{ \left(\sigma, \begin{pmatrix} \sigma \\ \sigma \end{pmatrix} \right) \mid \sigma \in \text{Traj} \right\} & \curvearrowright \text{---} \mapsto \left\{ \left(\begin{pmatrix} \sigma \\ \tau \end{pmatrix}, \sigma + \tau \right) \mid \sigma, \tau \in \text{Traj} \right\} \\ \text{---} \bullet \mapsto \{(\sigma, \bullet) \mid \sigma \in \text{Traj}\} & \circ \text{---} \mapsto \{(\bullet, 0)\} \\ \text{---} \boxed{r} \text{---} \mapsto \{(\sigma, \sigma \cdot r) \mid \sigma \in \text{Traj}\} & \text{---} \boxed{x} \text{---} \mapsto \{(\sigma, \sigma \cdot x) \mid \sigma \in \text{Traj}\} \end{array}$$

where \bullet is the only element of $k((x))^0$. The semantics of the reflected generators is symmetric, e.g. $\bullet \text{---}$ is mapped to $\{(\sigma, \bullet) \mid p \in k((x))\}$.

For space reasons, we refer the reader to [31] (and Remark 3 below) for more details on the semantics in terms of bi-infinite trajectories. Henceforth, we shall fix the choice of Laurent series $k((x))$ as the trajectories of choice.

Example 3. In Example 2, we presented the circuit c_2 as the composition of four sequential chunks. Their denotational semantics, following Definition 4, is below.

$$\begin{aligned} \llbracket (\bullet \text{---}; \text{---} \curvearrowright) \oplus \text{---} \rrbracket &= \{(\sigma_1, \begin{pmatrix} \tau_1 \\ \sigma_1 \end{pmatrix}) \mid \sigma_1, \tau_1 \in k((x))\} \\ \llbracket \text{---} \oplus (\curvearrowright \text{---}; \text{---} \bullet) \rrbracket &= \{(\begin{pmatrix} \tau_2 \\ \sigma_2 \\ \rho_2 \end{pmatrix}, \begin{pmatrix} \tau_2 \\ \sigma_2 + \rho_2 \\ \sigma_2 + \rho_2 \end{pmatrix}) \mid \sigma_2, \tau_2, \rho_2 \in k((x))\} \\ \llbracket (\text{---} \oplus \text{---} \boxed{x}) \oplus \text{---} \rrbracket &= \{(\begin{pmatrix} \tau_3 \\ \sigma_3 \\ \rho_3 \end{pmatrix}, \begin{pmatrix} \tau_3 \\ x \cdot \sigma_3 \\ \rho_3 \end{pmatrix}) \mid \sigma_3, \tau_3, \rho_3 \in k((x))\} \\ \llbracket (\curvearrowright \text{---}; \text{---} \bullet) \oplus \text{---} \rrbracket &= \{(\begin{pmatrix} \tau_4 \\ \sigma_4 \end{pmatrix}, \sigma_4) \mid \sigma_4, \tau_4 \in k((x))\} \end{aligned}$$

The composition in $\text{Rel}_{k((x))}$ of the four linear relations above is

$$\{(\sigma_1, \sigma_4) \mid \text{there exist } \sigma_2, \sigma_3, \tau_1, \dots, \tau_4, \rho_2, \rho_3 \text{ s.t. } \begin{cases} \tau_1 = \tau_2 = \sigma_2 = \tau_3 = \tau_4, \\ \sigma_2 + \rho_2 = \sigma_3, \quad x \cdot \sigma_3 = \tau_4 \\ \sigma_1 = \rho_2, \quad \sigma_2 + \rho_2 = \rho_3 = \sigma_4 \end{cases} \}$$

By simple algebraic manipulations one can check that the above systems of equations has a unique solution given by $\sigma_4 = \frac{1}{1-x}\sigma_1$. Since $\llbracket \cdot \rrbracket$ is a prop morphism and c_2 is the composition of the four chunks above, we obtain

$$\llbracket c_2 \rrbracket = \{(\sigma_1, \frac{1}{1-x} \cdot \sigma_1) \mid \sigma_1 \in k((x))\}.$$

This relation contains all pairs of streams that can occur on the left and on the right ports of c_2 . For instance if $\underline{1}, 0, 0 \dots$ is on the left, $\underline{1}, 1, 1 \dots$ is on the right.

For the other circuit of Example 2, namely c_1 , it is immediate to see that

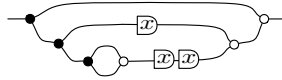
$$\llbracket c_1 \rrbracket = \{((1-x) \cdot \sigma_1, \sigma_1) \mid \sigma_1 \in k((x))\}$$

which is the same relation as $\llbracket c_2 \rrbracket$. In Example 4, we will prove the equivalence of the two circuits using equational reasoning, justified in the next section.

3.1 Interacting Hopf Algebras: the Equational Theory of Circuits

As witnessed by Example 3, different circuits (as morphisms of Circ) may have the same denotational semantics. It is thus natural to ask whether there is an equational theory which *axiomatises* denotational equality, i.e. it makes two circuits equal precisely when they denote the same relation. In fact, an equational theory exists, and it is remarkably modular.

The results in this section rely on various “polynomial-like” concepts: the ring of polynomials $k[x]$, its field of fractions $k(x)$, the ring $k\langle x \rangle$ of *rational* fractions of polynomials, formal power series $k[[x]]$, and its field of fractions, the field of Laurent series $k((x))$. We give a summary in Table 1. Note that polynomials $p \in k[x]$ are expressible in our syntax (4): for example, the polynomial $1+x+2x^2$ is captured by the following diagram:



$k[x]$	ring of polynomials	$\sum_0^n k_i x^i$ for some $n \in \mathbb{N}$
$k(x)$	field of polynomial fractions	$\frac{p}{q}$ for $p, q \in k[x]$ with $q \neq 0$
$k\langle x \rangle$	ring of rationals	$\frac{\sum_0^n k_i x^i}{\sum_0^m l_j x^j}$ with $l_0 \neq 0$
$k[[x]]$	ring of formal power series	$\sum_0^\infty k_i x^i$
$k((x))$	field of Laurent series	$\sum_d^\infty k_i x^i$ for some $d \in \mathbb{Z}$

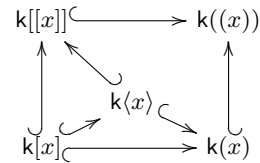


Table 1. Rings and fields over a field k (k_i and l_j range over k).

The equational theory, given in Figure 1, is comprised of the following ‘blocks’:

- In the first block, both the black and white structures form commutative monoids and comonoids, capturing properties of addition and copying.
- In the second block, the white monoid and black comonoid interact as a bimonoid. Bimonoids are one of two canonical ways that monoids and comonoids interact, as shown in [41].

- In the third and fourth block, both the black and the white monoid/comonoid pair form an extraspecial Frobenius monoid. The Frobenius equations (fr 1) and (fr 2) are a famous algebraic pattern which establishes a bridge between algebraic and topological phenomena, see [22, 40, 24]. The “extraspecial” refers to the last two equations on each line, two additional equations, the *special* equation (\bullet -sp, \circ -sp) and the *bone* equation (\bullet -bo, \circ -sp). The Frobenius equations, together with the special equation, are the other canonical pattern of interaction between monoids and comonoids identified in [41]. Together with the bone equation, the set of four equations characterises *corelations*, see [19, 64, 25].
- The equations in the fifth block are parametrised over $p \in k[x]$ and describe commutativity of $-\boxed{p}-$ with respect to the other operations, as well as multiplication and addition of scalars.
- The sixth block describes the interpretation of $-\overline{p}-$ as division by p , and shows how fractions are expressible in the circuit syntax.
- Finally, the last block expresses the possibility of changing the colour of “feedback loops”, modulo the insertion of the scalar -1 . In fact, it can be shown that these loops define a *compact closed structure* [39] on the category of circuits.

Remark 2. The axioms in Figure 1 are referred to as the theory of *Interacting Hopf Algebras* (\mathbb{H}). The name hints at the modular nature of the theory: the Frobenius monoid equations (fourth block), the fractions equations (sixth block) and the feedback equations (last block) can be seen as the result of combining the two Hopf algebras (the remaining blocks) via distributive laws of props [63, 18]. In fact, \mathbb{H} is modular at multiple levels: Hopf algebras themselves can be factorised in terms of distributive laws between monoids and comonoids [63, 18]. Another interesting observation is that the black (copying) and the white structure (addition) are totally symmetric, in the sense that \mathbb{H} is invariant wrt colour swapping. The symmetry between the the two colours is broken only by the 2-categorical structure of \mathbb{H} , see [12].

The theory \mathbb{H} is *sound and complete* for denotational equivalence of circuits.

Theorem 1. *For all circuits c, d in Circ , $\llbracket c \rrbracket = \llbracket d \rrbracket$ iff $c \stackrel{\mathbb{H}}{=} d$.*

The proof of this result relies on the modular structure of \mathbb{H} : exploiting Lack’s theory for composing props [41], completeness results for the sub-theories (*cf.* Theorem 3 below) are “put together” to achieve completeness for the whole \mathbb{H} , see [63, 18]. Also note that the same completeness theorem was independently obtained by Baez and Erbele [3] relying on a normal form argument.

Remark 3. It is worth mentioning that the denotational semantics in terms of bi-infinite trajectories $k^{\mathbb{Z}}$ (see Definition 4 and above) also enjoys a completeness results, in terms of an equational theory which is very close to \mathbb{H} . Most notably, the axiomatisation of bi-infinite trajectories lacks the axiom $-\boxed{p}-\overline{p}- = \text{—————}$, which is replaced by a weaker version. See [31] for the full details.

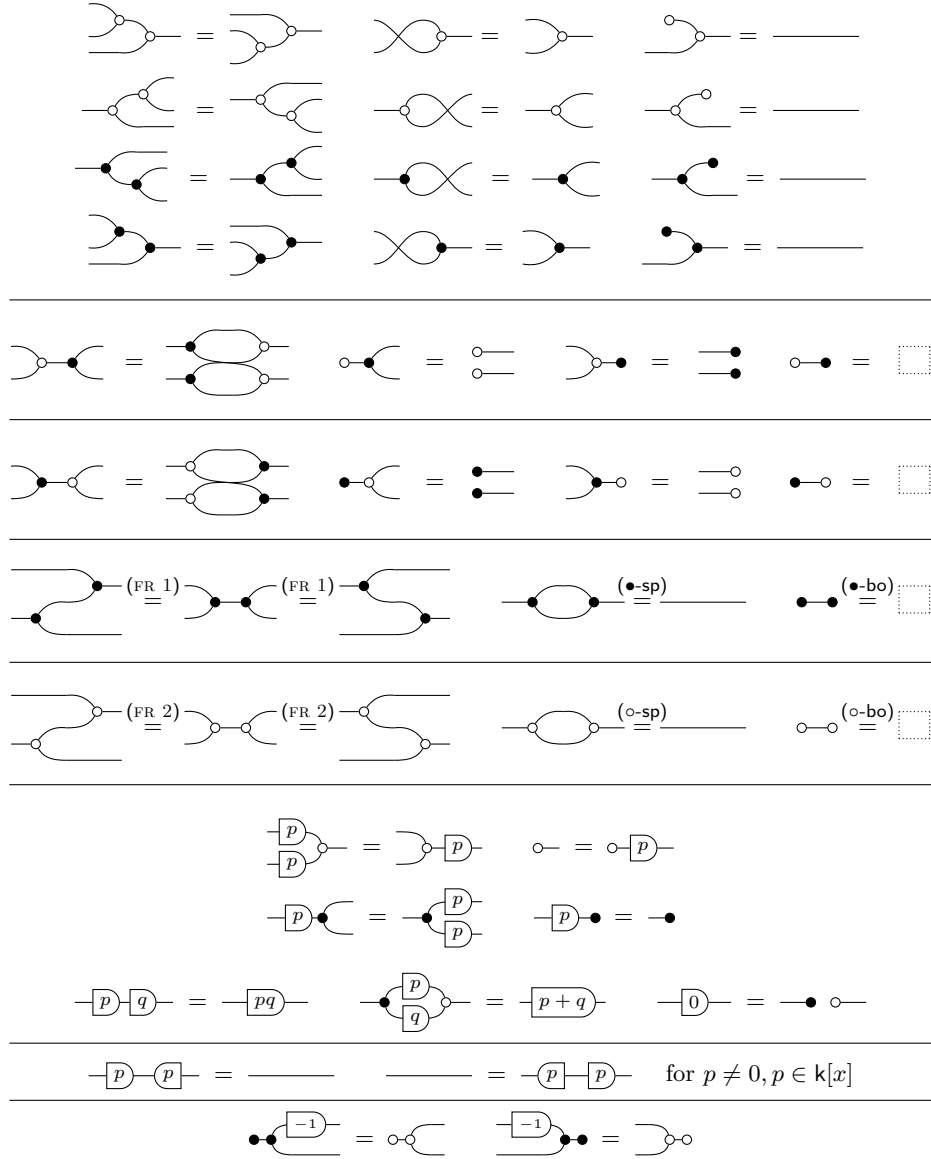
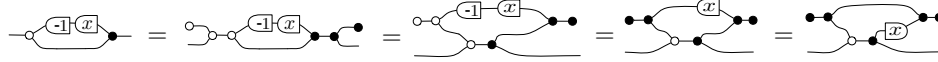


Fig. 1. Axioms of Interacting Hopf Algebras (\mathbb{H}).

Example 4. In Example 3, we have shown that the circuits c_1 and c_2 of Example 2 denotes the same relation. This equivalence can be proved by equational reasoning in \mathbb{H} , as the following derivation demonstrates:



Theorem 1 gives a partial account of Circ expressivity. Indeed, it states that $\llbracket \cdot \rrbracket : \text{Circ} \rightarrow \text{Rel}_{\mathbb{k}((x))}$ is *faithful*, but what about *fullness*? In other words, what property characterises the image of $\llbracket \cdot \rrbracket$? It turns out these are precisely the relations that are vector spaces over the field $\mathbb{k}(x)$ of polynomial fractions. Formally stated, this is Theorem 2 below, but first we recall two necessary concepts.

Definition 5 ($\text{LinRel}_{\mathbb{k}}$, $\text{Mat } \mathbb{k}$). *Given a field \mathbb{k} ,*

- $\text{LinRel}_{\mathbb{k}}$ is the sub-prop of $\text{Rel}_{\mathbb{k}}$ (Definition 3) with arrows those relations that are linear, namely an arrow $n \rightarrow m$ is a sub vector-space of \mathbb{k}^{n+m} .
- $\text{Mat } \mathbb{k}$ is the sub-prop of $\text{LinRel}_{\mathbb{k}}$ with arrows being just linear maps, namely an arrow $n \rightarrow m$ is a $m \times n$ -matrix over \mathbb{k} .

Writing \mathbb{H} for the quotient of Circ by $\stackrel{\mathbb{H}}{=}$, we may now state that

Theorem 2. *There is an isomorphisms of props between \mathbb{H} and $\text{LinRel}_{\mathbb{k}(x)}$.*

In other words, Theorem 2 says that the denotational semantics factors as

$$\begin{array}{ccc}
 \text{Circ} & \xrightarrow{\llbracket \cdot \rrbracket} & \text{Rel}_{\mathbb{k}((x))} \\
 \downarrow & & \uparrow \\
 \mathbb{H} & \xrightarrow{\cong} & \text{LinRel}_{\mathbb{k}(x)}
 \end{array}$$

We can provide similar results for the subprops of Circ introduced in Section 2. We write $\mathbb{H}\mathbb{A}$, $\mathbb{H}\mathbb{A}^{op}$ and $\mathbb{S}\mathbb{F}$ respectively for $\text{Circ}^{\rightarrow}$, Circ^{\leftarrow} and SF quotiented by $\stackrel{\mathbb{H}}{=}$.

Theorem 3. *There are isomorphisms of props between:*

1. $\mathbb{H}\mathbb{A}$ and $\text{Mat } \mathbb{k}[x]$;
2. $\mathbb{H}\mathbb{A}^{op}$ and $\text{Mat } \mathbb{k}[x]^{op}$;
3. $\mathbb{S}\mathbb{F}$ and $\text{Mat } \mathbb{k}\langle x \rangle$.

Remark 4 (Kleene theorem). The last point of Theorem 3 can be thought as an analogue of Kleene’s theorem for regular languages: it provides a syntactic characterisation of the *rational* behaviours. The relations denoted by $\mathbb{S}\mathbb{F}$ are particularly well behaved functions, since they do not actually require the full generality of Laurent series: any rational polynomial generates a finite power series, without the need for a “finite past.” The correspondence between (orthodox) signal-flow diagrams and rational matrices is well-known (see e.g. [54]): here we give a categorical, string-diagrammatic, account of this characterisation where notions of “input”, “output” and direction of flow are derivative.

We also mention that point 1 of Theorem 3 seems to be a folklore result, ubiquitously appearing in different flavours [42, 36, 52, 48, 41, 63].

4 Operational Semantics

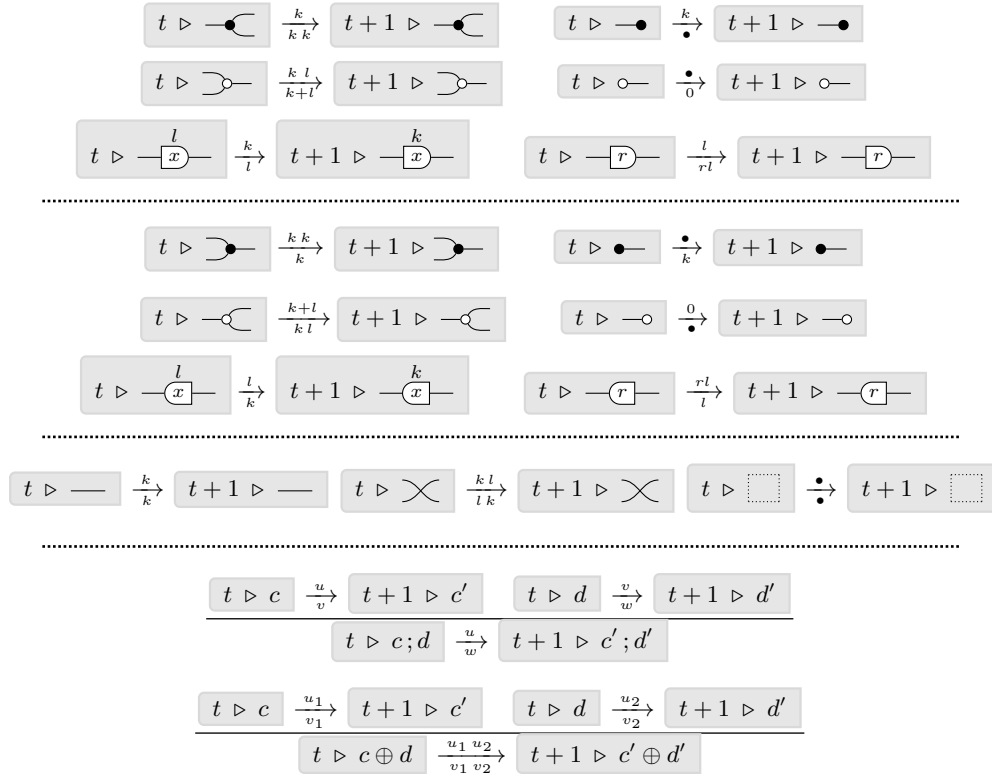


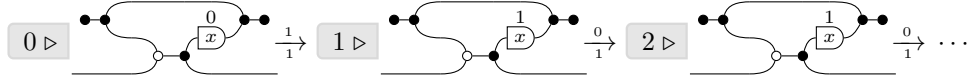
Fig. 2. Structural rules for operational semantics, with $p \in \mathbb{Z}$, k, l ranging over \mathbf{k} and u, v, w vectors of elements of \mathbf{k} of the appropriate size. The only vector of \mathbf{k}^0 is written as \bullet (as in Definition 4), while a vector $(k_1 \dots k_n)^T \in \mathbf{k}^n$ as $k_1 \dots k_n$.

As for programming languages, denotational semantics is just one way of giving formal meaning to circuit syntax. In this section we take a different perspective, and regard circuits as state-based machines, the step-by-step execution of which is the *operational semantics*.

The atomic components of the operational semantics will be transitions of shape $t \triangleright c \xrightarrow{\frac{v}{w}} t' \triangleright c'$. Here c and c' are *states*, that is, circuits augmented with information about which values $k \in \mathbf{k}$ are stored in each register $(\text{---}x\text{---})$ and $(\text{---}x\text{---})$ at that computation step. States are decorated with runtime contexts: t and t' are integers that—intuitively—indicate the time when the transition hap-

pens.⁴ Finally, the labels v and w are the values (k-vectors) observed respectively on the left and on the right interfaces of c , when transitioning to c' .

Example 5. For a concrete example, recall the circuit c_2 (Example 2). As shown in Example 3, the relation $\llbracket c_2 \rrbracket$ pairs $\underline{1}, 0, 0, \dots$ with $\underline{1}, 1, 1, \dots$. In the operational semantics, this is broken down into step-by-step observations, as follows:



At each step, the register \boxed{x} is annotated with its current value. One reads above trace as saying that, at time 0, value 1 is read both on the left and on the right; at time 1, value 0 is read on the left, and 1 on the right, and so on.

In the example above, the register is initialised at 0. We will assume this through the whole paper, as made explicit by the following definition.

Definition 6. *Let $c: n \rightarrow m$ be a circuit in Circ . The initial state c_0 of c is the one where all the registers store 0. A computation of c starting at time $t \leq 0$ is a (possibly infinite) sequence of transitions*

$$t \triangleright c_0 \xrightarrow[v_t]{v_t} t+1 \triangleright c_1 \xrightarrow[w_{t+1}]{v_{t+1}} t+2 \triangleright c_2 \xrightarrow[w_{t+2}]{v_{t+2}} \dots \quad (6)$$

It is important to remark that the rules in Fig. 2 define the valid transitions for any circuit, inductively on the syntax of Circ . The definitions for the generators reflect the intuition of Section 2; for instance, $\bullet \curvearrowright$ acts as a *copier*, because when k is observed on the left then two copies of k are observed on the right. Also, all the generators are stateless except for the registers \boxed{x} and \boxed{x} . For \boxed{x} , the idea is that whenever it receives a value k on the left, it releases the currently stored value l on the right, and k becomes the newly stored value. Symmetrically, in the definition for \boxed{x} (and, more generally, any generator of Circ^{\leftarrow}) it is helpful to think of signals as coming from the right, and exiting from the left.

This description hints at a potential issue. On the one hand, the purpose of providing an operational semantics is to describe step-by-step evolution of circuits as executable machines. On the other hand, we allow for the composition of circuits, such as those of $\text{Circ}^{\rightarrow}$ and of Circ^{\leftarrow} , in which signals intuitively flow in different directions. What is the computational meaning of such composites?

As seen in Example 5, sometimes it is possible to construct computationally meaningful circuits with a mix of $\text{Circ}^{\rightarrow}$ - and Circ^{\leftarrow} -components: c_2 has a clear left-to-right directionality, as the Circ^{\leftarrow} -components ($\bullet \curvearrowright$ and $\curvearrowright \bullet$) only contribute to a feedback loop. Other circuits exhibit a more puzzling behaviour, for example:

⁴ Note that, being an integer, time may be negative — as we shall see in Example 6, some executions must start in the past.

Example 6. In $-(\boxed{x})-(\boxed{x})-$, both interfaces seem to act as outputs. The equational theory \mathbb{H} equates it with $-$, and thus they have the same denotational semantics. However, their operational behaviour is subtly different. Indeed, whereas for any sequence $a_i \in \mathbf{k}$, $-$ admits the computation

$$\boxed{0} \triangleright - \xrightarrow[a_0]{a_0} \boxed{1} \triangleright - \xrightarrow[a_1]{a_1} \boxed{2} \triangleright - \xrightarrow[a_2]{a_2} \dots \quad (7)$$

The circuit $-(\boxed{x})-(\boxed{x})-$ admits a similar computation, but we must begin at time $t = -1$ in order to first “load” the registers with a_0 :

$$\boxed{-1} \triangleright - \xrightarrow[0]{0} \boxed{0} \triangleright - \xrightarrow[a_0]{a_0} \boxed{1} \triangleright - \xrightarrow[a_1]{a_1} \dots \quad (8)$$

The circuit $-(\boxed{x})-(\boxed{x})-$, which again is equal to $-$ in \mathbb{H} , has yet a different operational behaviour. Although every computation of $-$ can be reproduced, $-(\boxed{x})-(\boxed{x})-$ admits additional, problematic computations. Indeed, consider

$$\boxed{0} \triangleright - \xrightarrow[1]{0} \boxed{1} \triangleright - \quad (9)$$

at which point no further transition is possible—the circuit can deadlock.

At a deeper level, Example 6 demonstrates that the operational semantics is not meant to be executable for all circuits: the rule for sequential composition implicitly quantifies existentially on the middle value v , resulting in potentially unbounded non-determinism. Reaching a satisfactory understanding of this phenomenon is the subject of the next section. As a preliminary observation, it is worth observing that, if one restricts to *infinite* computations, then the mismatch outlined in Example 6 disappears, and we have a perfect correspondence between operational and denotational equivalence.

In order to state this result, we introduce a preliminary notion. As customary in control theory, we consider *trajectories*: traces that possibly start in the past.

Definition 7. *To any infinite computation as in (6) we can associate a \mathbb{Z} -indexed sequence $\sigma : \mathbb{Z} \rightarrow \mathbf{k}^n \times \mathbf{k}^m$, called a trajectory, as follows:*

$$\sigma(i) = \begin{cases} (u_i, v_i) & \text{if } i \geq t, \\ (0, 0) & \text{otherwise.} \end{cases} \quad (10)$$

Note that σ is finite in the past, i.e., for which $\exists j \in \mathbb{Z}$ such that $\sigma(i) = (0, 0)$ for $i \leq j$.

We call operational semantics the set $\langle c \rangle$ of trajectories given by the infinite computations of c .

Remark 5. We purposefully used the same term “trajectories” as in the denotational semantics: indeed, there is a close connection between computations that start in an initial state and Laurent series, which we encountered in Section 3.

Indeed the sequence of observations at any port is clearly a Laurent series – any computation can be trivially (i.e. with 0 observations) continued infinitely into the past, as reflected by (10) in Definition 6. One could generalise the notion of computation so that one would not need to start in the initialised state: in that case, the corresponding notion of trajectory 10 could be a bona-fide bi-infinite sequence, i.e. with σ possibly infinite in the past. See [31, 30] for more details.

Theorem 4 (Operational-Denotational correspondence). *For all circuits c, d of Circ, $\langle c \rangle = \langle d \rangle$ if and only if $\llbracket c \rrbracket = \llbracket d \rrbracket$.*

Example 7. Consider (7) and (8) in Example 6. According to (10) both are translated into the trajectory σ mapping $i \geq 0$ to (a_i, a_i) and $i < 0$ into $(0, 0)$. More generally, it holds that $\langle \text{---} \rangle = \langle \text{---} \oplus \text{---} \rangle$. Note the two circuits would be distinguished when looking at their finite traces— compare (7) with (8). However, by Theorem 4, $\langle \text{---} \rangle = \langle \text{---} \oplus \text{---} \rangle$ also holds. Indeed, problematic computations, like (9), are all finite and, by definition, do not give rise to any trajectory.

5 Realisability

In light of Example 6, one may ask how common deadlock situations (as the one in (9)) and computations that need to start in the past (as in (8)) are. It turns out these issues are avoided when considering the operational semantics of the sub-class SF of circuits that adhere to the classical notion of signal flow graphs.

Lemma 1. *Let c be a circuit diagram in SF.*

1. *There are no deadlocks - for every $t \in \mathbb{Z}$ and state c_t , there exists a transition*

$$t \triangleright c_t \xrightarrow[w_t]{v_t} t + 1 \triangleright c_{t+1} .$$

2. *Every computation with only trivial observations in the past can be started at time 0. That is, for any $k \in \mathbb{N}$, if*

$$\boxed{-k \triangleright c_0} \xrightarrow[0]{0} \boxed{-k + 1 \triangleright c_1} \xrightarrow[0]{0} \cdots \xrightarrow[0]{0} \boxed{0 \triangleright c_k} \xrightarrow[w_k]{v_k} \boxed{0 \triangleright c_{k+1}} \xrightarrow[w_{k+1}]{v_{k+1}} \cdots$$

then also

$$\boxed{0 \triangleright c_0} \xrightarrow[w_k]{v_k} \boxed{1 \triangleright c_{k+1}} \xrightarrow[w_{k+1}]{v_{k+1}} \boxed{2 \triangleright c_{k+2}} \xrightarrow[w_{k+2}]{v_{k+2}} \cdots$$

The key insight behind this result is that, in SF, the existential quantification in sequential circuit composition becomes deterministic, subject to a choice of inputs at each step of evaluation. Therefore, circuits in SF provide a proper operational realisation of the relation behaviour that they denote. What can we say about the other behaviours denoted in Circ? Can they be executed somehow? Do they have a proper operational realisation?

Below we shall see that the answer to the last questions is positive, in fact, within the equational theory of \mathbb{H} , Circ is nothing but a “jumbled up” version of SF : more precisely, while every circuit in SF has inputs on the left and outputs on the right, for every circuit in Circ there is a way of partitioning its left and right ports into “inputs” and “outputs”, in the sense that appropriate rewiring yields an \mathbb{H} -equal circuit in SF .

We begin by giving a precise definition of what we mean by “jumbling up” the wires of a circuit. First, for each $n, m \in \mathbb{N}$, we define circuits $\eta_n : n \rightarrow 1+1+n$ and $\epsilon_m : 1+1+m \rightarrow m$ in Circ as illustrated below.

$$\eta_n := \begin{array}{c} \bullet \quad \bullet \\ \text{---} \quad \text{---} \\ \text{---} \quad \text{---} \\ \boxed{n} \end{array} \quad \epsilon_m := \begin{array}{c} \text{---} \quad \text{---} \\ \bullet \quad \bullet \\ \text{---} \quad \text{---} \\ \boxed{m} \end{array}$$

Next, we define the families of operators $L_{n,m} : \text{Circ}[n+1, m] \rightarrow \text{Circ}[n, 1+m]$ and $R_{n,m} : \text{Circ}[n, 1+m] \rightarrow \text{Circ}[1+n, m]$ as follows: for any circuit $c \in \text{Circ}[n+1, m]$,

$$L_{n,m}(c) = \eta_n ; (id_1 \oplus c) \quad \left(\begin{array}{c} \text{---} \quad \text{---} \\ \bullet \quad \bullet \\ \text{---} \quad \text{---} \\ \boxed{c} \\ \text{---} \quad \text{---} \\ \boxed{n} \quad \boxed{m} \end{array} \right)$$

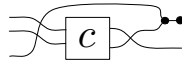
and, for any circuit $d \in \text{Circ}[n, m+1]$

$$R_{n,m}(d) = (id_1 \oplus d) ; \epsilon_m \quad \left(\begin{array}{c} \text{---} \quad \text{---} \\ \text{---} \quad \text{---} \\ \bullet \quad \bullet \\ \text{---} \quad \text{---} \\ \boxed{d} \\ \text{---} \quad \text{---} \\ \boxed{n} \quad \boxed{m} \end{array} \right)$$

Definition 8. A circuit $c_2 \in \text{Circ}[n_2, m_2]$ is a rewiring of $c_1 \in \text{Circ}[n_1, m_1]$ when c_2 can be obtained from c_1 by a combination of the following operations:

- (i) application of $L_{n,m}$, for some n and m ,
- (ii) application of $R_{n,m}$, for some n and m ,
- (iii) post-composition with a permutation,
- (iv) pre-composition with a permutation.

Permutations are needed to rewire an arbitrary—i.e. not merely the first—port on each of the boundaries. For instance, they allow to rewire the second port on the right as the third on the left in the circuit $c : 2 \rightarrow 2$ below:

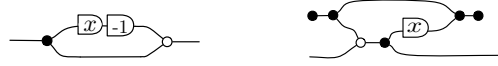


At the semantics level, a rewiring denotes an isomorphism between a subspace of type $k(x)^n \times k(x)^m$ and one of type $k(x)^i \times k(x)^j$ where $n+m = i+j$. For instance, for any circuit c , $\llbracket c \rrbracket \subseteq k(x)^{n+1} \times k(x)^m$ is isomorphic to $\llbracket L_{n,m}(c) \rrbracket \subseteq k(x)^n \times k(x)^{m+1}$ as a subspace of $k(x)^{n+m+1}$.

Theorem 5 (Realisability). For every circuit $c \in \text{Circ}$, there exists $d \in \text{SF}$ such that c is \mathbb{H} -equivalent to some rewiring of d .

The above theorem guarantees that an input-output partition of the ports of any circuit in Circ always exists. Note that such a partition is not unique, as illustrated by the following example.

Example 8. The circuit c_1 of Example 2 is equivalent to the rewiring of two different signal flow graphs, illustrated below.



Indeed the rightmost circuit above is c_2 of Example 2 which is \mathbb{H} -equivalent to c_1 . The leftmost circuit is $R_{0,1}(L_{0,1}(c_1); \infty)$. Intuitively, the rightmost circuit above corresponds to choosing the leftmost port of c_1 to be the input and the rightmost to be the output; in the rightmost diagram we do the opposite the choice: the leftmost port of c_1 is the output and the rightmost is the input.

5.1 Compositionality vs Directionality of Signal Flow

The fact that the input-output partition is not unique corresponds to the fact in some circuits there is more than one way of orienting flow. This allows us to crystallise what we consider to be a central methodological contribution of the compositional approach: since it is only by forgetting the input-output distinction that the algebra \mathbb{H} of signal flow is revealed, and signal flow graphs can be given a compositional semantics. The notions of input and output cannot, therefore, be considered as primitive; they are, rather, derived notions. This is different from classical approaches – for example Mason [45] emphasises inputs, outputs and flow directional as a central feature:

“flow graphs differ from electrical network graphs in that their branches are directed. In accounting for branch directions it is necessary to take an entirely different line of approach...”

The compositional approach is therefore closely connected to the approach of Willems [61] and *behavioural control theory*, which emphasises a relational treatment of observable behaviour. Willems himself railed [62] against including the notions of inputs, outputs and directionality in fundamental mathematical descriptions of interconnected systems:

“Adding a signal flow direction is often a figment of one’s imagination, and when something is not real, it will turn out to be cumbersome sooner or later. [...] The input/output framework is totally inappropriate for dealing with all but the most special system interconnections. [Inputs/outputs] often needlessly complicates matters, mathematically and conceptually.”

Inputs, outputs and flow directionality *are* crucial for when one wants to an implementation of a behaviour—i.e. an executable circuit—as clarified at the beginning of this section. Since any circuit in *Circ* denotes the same relation as a rewiring of a signal flow graph, then all the denoted behaviours can be properly realised. This explains the name “Realisability” for Theorem 5.

Circuits, their algebraic theory, and their compositional semantics can thus be considered a bona fide process algebra of signal flow, and serve both as a language for *specifications* and for (executable) *implementations*. In fact, the language lends itself to formal methods techniques such as *refinement*, see [10].

6 The Affine Extensions

Engineers usually do not distinguish between linear and *affine* systems, as the numerical methods to study the two are substantially the same. From our perspective however, such distinction is important since, in order to express affine behaviours, we need to extend the calculus of signal flow graphs, as well as the main results illustrated so far. Such extension turns out to be extremely interesting because on the one hand, it enables the modelling of systems with richer patterns of behaviour, like current and voltage sources in Section 7 or mutual exclusion in Section 8; on the other hand, it allows to define *contextual equivalence* and turn Theorem 4 in a proper *full abstraction result* (Theorem 8).

As usual, we start with the syntax. The syntactic prop ACirc is obtained by extending the grammar in (4) with an extra constant, \dashv , having 0 interfaces on the left and 1 on the right.

The denotational semantics $\llbracket \cdot \rrbracket : \text{ACirc} \rightarrow \text{Rel}_{k((x))}$ extends Definition 4 with

$$\dashv \mapsto \{(\bullet, 1)\}.$$

where $1 \in k((x))$ denotes the Laurent series $0, \underline{1}, 0, 0, \dots$. Thus, intuitively, \dashv emits such streams on the right port.

For the equational theory, we need to identify which axioms govern the behaviour of the extra connector \dashv . It turns out [13] that following three equations suffice.

$$\begin{array}{c} \text{---} \bullet \text{---} \\ \text{---} \end{array} \stackrel{(\text{dup})}{=} \begin{array}{c} \text{---} \bullet \text{---} \\ \text{---} \end{array} \quad \begin{array}{c} \text{---} \bullet \text{---} \\ \text{---} \end{array} \stackrel{(\text{del})}{=} \begin{array}{c} \text{---} \bullet \text{---} \\ \text{---} \end{array} \quad \text{---} \dashv \text{---} \stackrel{(\emptyset)}{=} \text{---} \bullet \bullet \text{---} \quad (11)$$

The first two say that \dashv can be deleted and copied by the comonoid structure, just like \circ . The third equation is justified by the possibility of expressing the empty set, by, for example,

$$\llbracket \dashv \circ \rrbracket = \{(\bullet, 1)\}; \{(0, \bullet)\} = \emptyset. \quad (12)$$

Since for any R and S in $\text{Rel}_{k((x))}$, $\emptyset \oplus R = \emptyset \oplus S = \emptyset$, composing or taking the monoidal product of \emptyset with any relation results in \emptyset ; \emptyset is thus analogous to logical false. Indeed we can use equation (\emptyset) to derive the following lemma.

Lemma 2. *For any two circuit $c, d: k \rightarrow l$ of ACirc,*

$$\begin{array}{c} \dashv \circ \\ k \text{---} \boxed{c} \text{---} l \end{array} \stackrel{\text{a} \parallel \text{H}}{=} \begin{array}{c} \dashv \circ \\ k \text{---} \boxed{d} \text{---} l \end{array}$$

In the above lemma and hereafter, $\stackrel{\text{allH}}{=}$ stands for the smallest congruence on circuits in ACirc containing the axioms in Figure 1 and those in (11). Moreover, we call AIH the quotient of ACirc by $\stackrel{\text{allH}}{=}$.

Theorem 6. *For all circuits c, d in ACirc, $\llbracket c \rrbracket = \llbracket d \rrbracket$ iff $c \stackrel{\text{allH}}{=} d$.*

To characterise the expressivity of ACirc, we need to introduce *affine* relations [13].

Definition 9 (A Rel_R). *Let R be a semiring. For $R, S \subseteq R^n$, their Minkowski sum is defined as $R + S = \{u + v \mid u \in R \text{ and } v \in S\}$. A set $R \subseteq R^n$ is said to be an affine subspace if there exist finite $B, D \subseteq R^n$ such that $R = \bigcup_{v \in B} \{v\} + \langle D \rangle$. Elements of B are called base points and those of D the directions.*

A R -affine relation $R: n \rightarrow n$ is an affine subspace $R \subseteq R^n \times R^m$. The prop ARel_R is the sub-prop of Rel_R (Definition 3) with arrows being affine relations.

When R is a field, the above definition simplifies: an affine subspace of R^n is a subset $R \subseteq R^n$ which is either empty or there exists a vector $u \in R^n$ and a linear subspace L of R^n such that $R = u + L := \{u + v \mid v \in L\}$. When R is the field of fraction of polynomials, $k(x)$, $\text{ARel}_{k(x)}$ exactly characterises the expressivity of ACirc.

Theorem 7. *There is an isomorphisms of PROPs between AIH and $\text{ARel}_{k(x)}$.*

The proof of this result uses a normal form argument, building on the characterisation of Theorem 2 [13].

6.1 Full Abstraction

A first payoff of the affine extension is the possibility of formulating a notion of *contextual equivalence* for circuit diagrams, and consequently a *full abstraction* result.

To this aim, first we extend the operational semantics to the syntax ACirc. This amounts to augmenting the rules in Figure 2 with

$$0 \triangleright \vdash \xrightarrow{\bullet_1} 1 \triangleright \vdash \qquad t \triangleright \vdash \xrightarrow{\bullet_0} t+1 \triangleright \vdash \quad (t \neq 0)$$

The behaviour of the affine generator \vdash depends on the time: when $t = 0$, it emits 1 on the right, otherwise it emits 0. It is easy to see that $\langle \vdash \rangle$ contains only the trajectory (see Definition 7) mapping time 0 into $(\bullet, 1)$ and all the other times $t \in \mathbb{Z}$ into $(\bullet, 0)$. The operational behaviour of \dashv is symmetrical. Observe that the behaviour of all other generators in Figure 2 is time-independent.

Example 9. Recall from (12) that $\llbracket \vdash \circ \rrbracket = \emptyset$. The same happens with the operational semantics: $\vdash \circ$ has no possible transition at $t = 0$, since at that time \vdash must emit a 1 and \circ can only synchronise on a 0. Instead, the circuit \square can always perform an infinite computation $t \triangleright \square \xrightarrow{\bullet} t+1 \triangleright \square \xrightarrow{\bullet} \dots$, for any $t \leq 0$. It is worth observing that for all c , $\square \oplus c$ can perform the same computations of c , while $\vdash \oplus c$ cannot ever make a transition at time 0.

Roughly speaking, the computations of $\dashv\circ$ and \square mirror images of the two possible denotations: there are only two $0 \rightarrow 0$ arrows in $\text{Rel}_{k((x))}$: the empty relation and the identity one. The former intuitively corresponds to logical false, as we said earlier; the second to logical truth.

Definition 10. For a circuit $c \in \text{ACirc}[0, 0]$ we write $c \uparrow$ if c can perform an infinite computation and $c \not\uparrow$ otherwise. For instance $\square \uparrow$, while $\dashv\circ \not\uparrow$.

We take the predicate \uparrow as our basic operational observation. To be able to make observations about arbitrary circuits we need to introduce an appropriate notion of context. Roughly speaking, contexts for us are circuits in $\text{ACirc}[0, 0]$ with a hole into which we can plug another circuit. Since ours is a variable-free presentation, “dangling wires” assume the role of free variables [33]: restricting to contexts in $\text{ACirc}[0, 0]$ is therefore analogous to considering *ground* contexts—i.e. contexts with no free variables—a standard concept of programming language theory. For more details on contexts, we refer the reader to [14].

With this setup, given a circuit $c \in \text{ACirc}[n, m]$, we can insert it into a context $C[-]$ and observe the possible outcome: either $C[c] \uparrow$ or $C[c] \not\uparrow$. This naturally leads us to contextual equivalence and the formulation of full abstraction.

Definition 11. Given $c, d \in \text{ACirc}[n, m]$, we say that they are contextually equivalent, written $c \equiv d$, if for all contexts $C[-]$,

$$C[c] \uparrow \text{ iff } C[d] \uparrow.$$

Theorem 8 (Full abstraction). $c \equiv d$ iff $c \stackrel{\text{obH}}{=} d$.

Example 10. Recall from Example 6, the circuits --- and $\text{---}\square\text{---}$. Take the context $C[-] = c_\sigma; -; c_\tau$ for $c_\sigma \in \text{ACirc}[0, 1]$ and $c_\tau \in \text{ACirc}[1, 0]$. Assume that c_σ and c_τ have a single infinite computation. Call σ and τ the corresponding trajectories. If $\sigma = \tau$, both $C[\text{---}]$ and $C[\text{---}\square\text{---}]$ would be able to perform an infinite computation. Instead if $\sigma \neq \tau$, none of them would perform any infinite computation: --- would stop at time t , for t the first moment such that $\sigma(t) \neq \tau(t)$, while $C[\text{---}\square\text{---}]$ would stop at time $t + 1$.

Now take as context $C[-] = \bullet\text{---}; -; \text{---}\bullet$. In contrast to c_σ and c_τ , $\bullet\text{---}$ and $\text{---}\bullet$ can perform more than one single computation: at any time they can nondeterministically emit any value. Thus every computation of $C[\text{---}] = \bullet\text{---}\bullet$ can *always* be extended to an infinite one, forcing synchronisation of $\bullet\text{---}$ and $\text{---}\bullet$ at each step. For $C[\text{---}\square\text{---}] = \bullet\text{---}\square\text{---}\bullet$, $\bullet\text{---}$ and $\text{---}\bullet$ may emit different values at time t , but the computation will get stuck at $t + 1$. However, our definition of \uparrow only cares about whether $C[\text{---}\square\text{---}]$ can perform an infinite computation. Indeed it can, as long as $\bullet\text{---}$ and $\text{---}\bullet$ consistently emit the same value at each time step.

If we think of contexts as tests, and say that a circuit c passes test $C[-]$ if $C[c]$ perform an infinite computation, then our notion of contextual equivalence is *may-testing* equivalence [27]. From this perspective, --- and $\text{---}\square\text{---}$ are not

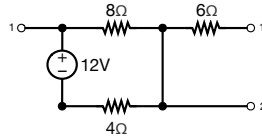
must equivalent, since the former must pass the test $\bullet - ; - ; - \bullet$ while $- \boxed{x} - \boxed{x} -$ may not.

It is worth to remark here that the distinction between may and must testing cease to make sense for circuits equipped with a proper flow directionality and thus a deterministic, input-output, behaviour. It is indeed possible to have for ACirc a weaker form of Realisability (Theorem 5) that we avoid to illustrate in this survey. We refer the interested reader to Sections 5 and 6 of [14].

7 Electrical Circuits

A main advantage of adopting an abstract and very elementary circuit syntax is that signal flow graphs become just one of the computational models that can be studied therein. In this section, we will focus on electrical circuits, showing how they can be modelled within ACirc. In the next section, we will sketch how yet a different model, Petri nets, can be given a similar account.

Elementary electrical engineering focusses on open linear circuit analysis. Such circuits may include voltage $(\overset{k}{- \oplus -})$ and current sources $(\overset{k}{- \ominus -})$, resistors $(\overset{k}{- \text{zigzag} -})$, inductors $(\overset{k}{- \text{m} -})$, capacitors $(\overset{k}{- \text{||} -})$, junctions (filled nodes) and open terminals (unfilled nodes). An example is illustrated below.



We may encode these systems syntactically as the morphisms of a prop ECirc, freely generated by the signature

$$\Sigma = \{ \text{---}, \text{---}, \text{---}, \text{---} \} \cup \{ \overset{k}{\text{zigzag}}, \overset{k}{\text{m}}, \overset{k}{\text{||}}, \overset{k}{- \oplus -}, \overset{k}{- \ominus -} \mid k \in R_+ \} \quad (13)$$

where the parameter k ranges over the non-negative reals. Arrows $m \rightarrow n$ of ECirc represent open linear electrical circuits with m open terminals on the left and n open terminals on the right.

Next, we provide an encoding of electrical circuits into circuit diagrams. This amounts to define a function $\mathcal{J}(-)$ that translates arrows in $\text{ECirc}[n, m]$ into arrows $\text{ACirc}[2n, 2m]$. The 2 is needed because each electrical wire carries 2 quantities (current and voltage) and thus its represented by two wires in ACirc. For each generator in Σ , Figure 3 provides its translation into a circuit of ACirc. The translation for arbitrary circuits follows inductively by the one of generators.

Figure 4 illustrates some well-known properties of electric components in series and parallel. The main payoff of working in ACirc is that now these properties can be now proved equationally by using the theory AIH. For instance, the

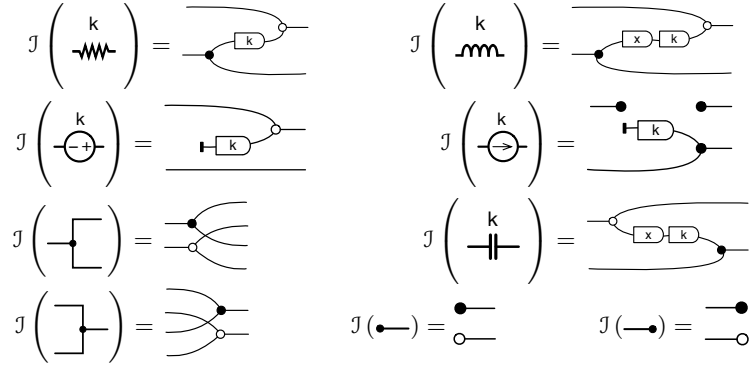


Fig. 3. Compositional encoding of open electrical circuits.

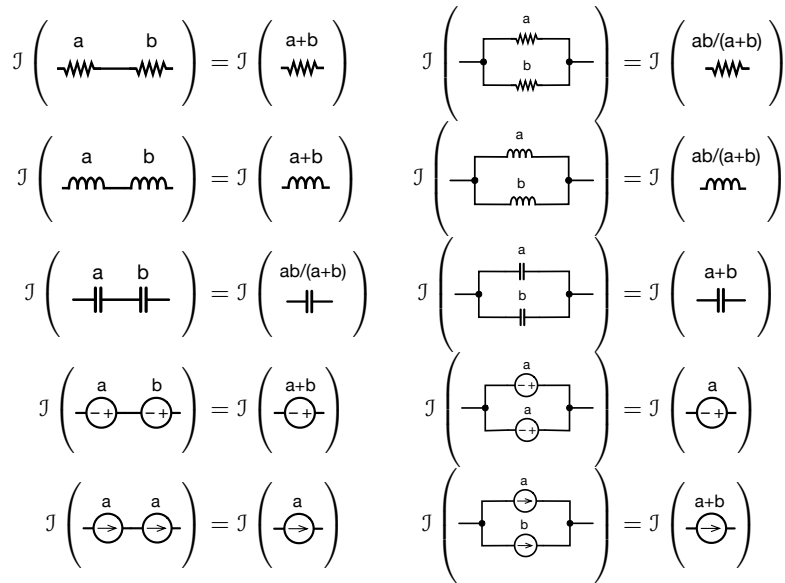
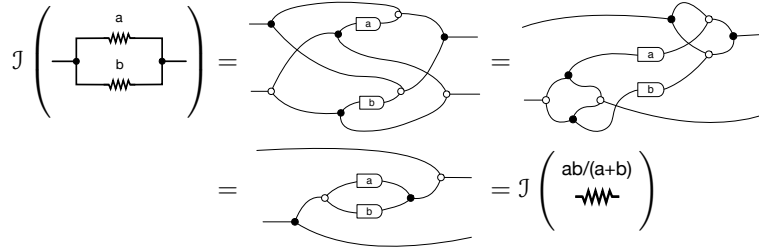
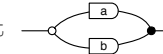
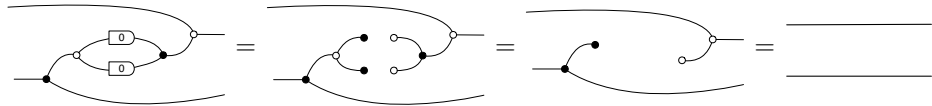


Fig. 4. Properties of resistors, inductors, capacitors, voltage and current sources in sequence (left) and parallel (right)

equivalence in the first row of the second column (resistors in parallel) can be proved as follows.

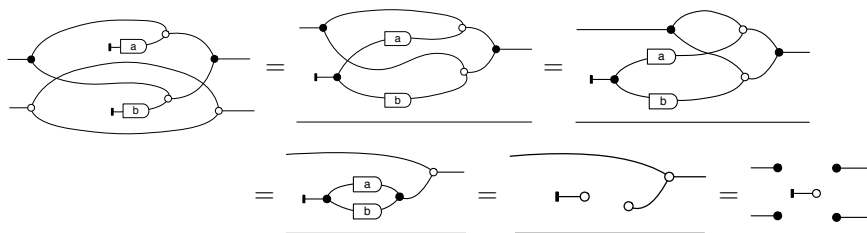


Remark 6. The circuit , which appears in the last step above, means $\frac{1}{\frac{1}{a} + \frac{1}{b}} = \frac{ab}{a+b}$ when both $a \neq 0$ and $b \neq 0$. Note, however, that it is more general than the traditional $\frac{ab}{a+b}$, e.g. it is well-defined even when both $a = b = 0$. In that case, parallel zero resistors reduce, as expected, to a wire:



Similarly, the expression behaves as expected if one substitutes *infinite* resistances, which in graphical linear algebra is “1/0”, i.e. $\text{---} \circ \text{---}$. The same holds for inductors in series and capacitors in parallel.

Remark 7. Observe that in in the rightmost column of the fourth row of Figure 4, the two parallel voltage source must have the same voltage a . Indeed, in engineering literature, parallel voltage sources of different voltages are disallowed. It is nonetheless interesting to see what happens in the semantics.



This, as we have seen, is the way of expressing the empty relation in the affine calculus. The same holds for current source in series (first column, last row).

Remark 8. For more details about the encoding of electrical circuits in affine circuit diagrams, the reader may refer to [13]. A similar semantics was given by Baez and Coya [37, 26], building on the work of Baez, Erbele and Fong [3, 6], and Rosebrugh, Sabadini and Walters [53]. However, such works only consider *passive* linear circuits, namely circuits without voltage and current sources.

8 From Control Theory to Concurrency

We have shown that the calculus of signal flow diagrams admits an axiomatisation of a significant family of behaviours (linear dynamical systems), capturing a well-known pre-existing combinatorial model (signal flow graphs). The approach is compositional, allowing for the syntactic representation of open systems and emphasising their algebraic properties, but at the same time is graphical, emphasising their connection topology and combinatorial properties.

Historically, putting different emphasis on these aspects has led to different research threads in the analysis of *concurrent systems*. On the one hand, we have the algebraic approach put forward by *process calculi* [47, 56, 35]. On the other hand, there is the tradition of graphical models of concurrent behaviour, such as *Petri nets* (see e.g. [50]).

It thus seems that the calculus signal flow diagrams may act as a middle ground between the perspectives offered by process calculi and by graphical models. This led us [11] to analyse concurrent systems with the same diagrammatic approach used for the analysis of linear behaviour in signal flow theory.

What is most striking is that, passing from linear to concurrent behaviour, the setup may remain essentially unaltered: one can use the same generators of the syntax (4), and the only significant change is modelling their behaviour via a different set of signals, passing from a field k to the semiring of natural numbers \mathbb{N} .

In order to explain this point, we illustrate two examples from [13] and [11].

$$c_1 := \text{---} \circ \text{---} \quad c_2 := \text{---} \bullet \text{---} \bullet \text{---} \quad (14)$$

In the (affine) calculus of signal flow diagrams, the behaviour of c_1 is not particularly interesting. It is the relation

$$\left\{ \left(\bullet, \begin{pmatrix} \sigma \\ \tau \end{pmatrix} \right) \mid \sigma + \tau = 1 \right\}$$

where σ and τ are arbitrary trajectories over a field k . Switching the signal space from k to \mathbb{N} , gives us something much more relevant, namely *mutual exclusion*: the circuit has only two possible behaviours:

$$\left\{ \left(\bullet, \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right), \left(\bullet, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right) \right\}.$$

Indeed there are only those two possible solutions to the equation $\sigma + \tau = 1$ when σ and τ are trajectories over \mathbb{N} . The fact that signals are in \mathbb{N} , and thus cannot be negative, allows us to interpret them as *resources*: they can be shared and consumed and their absence may inhibit certain behaviours.

This become even more clear when considering the circuit c_2 . Its behaviour as signal flow diagram is trivial: it is the full relation $\text{---} \bullet \text{---} \bullet \text{---}$, relating any input to any output. To see this it is convenient to think operationally: assume some

value $s \in k$ is currently in the register. Now, given input k on the left, in order to output k' on the right, one just needs to find k'' such that $k' + k'' = s$, which is always possible by taking $k'' = s - k'$.

Note that this is not the case, when k', k'', s range over \mathbb{N} : the output k' should be *smaller* than the content of the register s . If one interprets these quantities as *tokens*, the circuit c_2 is exactly a *place* of a Petri nets: from a place containing s tokens can be taken a number of tokens $k' \leq s$ and can be inserted an arbitrary number of tokens k . After these operations, the place would contain $k + (s - k')$, i.e., $k + k''$, tokens.

These observations have been crystallised in the *resources calculus* [51, 11]: the syntax is the same as for the calculus of signal flow diagrams, but the signal universe is fixed to be \mathbb{N} . This switch forces to move from *linear* to *additive* relations and, consequently, to change the axiomatisation [11]. In [13], it is shown that the algebra of stateless connectors [20] can be easily encoded into the (affine extension) of the resource calculus. An extended study of Petri nets within the resource calculus is illustrated in [11]. This provides an insightful understanding as Petri nets as linear dynamical systems (but over \mathbb{N}) as well as an elegant compositional operational semantics. The denotational semantics instead appears to be challenging and surely deserves further investigations.

9 This Research and IFIP AICT 600 - IFIP 60 Year Festivity Issue

The research described in this paper intersects with the interests of two TC1 working groups: *IFIP-WG 1.3 Foundations of System Specification* and *IFIP-WG 1.8 Concurrency Theory*. It also touches on TC2 topics, being relevant to *IFIP-WG 2.2 Formal Description of Programming Concepts*. First, signal flow graphs are a simple, yet pervasive, model of computation. The 2-dimensional approach described here is flexible enough to enable the string diagrammatic syntax to express both formal specifications as well as implementations. The axiomatic characterisation means that the former can be transformed in a principled way to the latter. Further, the operational semantics we give is a kind of concurrent process algebra, and—in the affine extension—we explore ramifications for the study of the semantics of Petri nets, a classical model of concurrency. Finally, the emphasis on the interaction between denotational and operational approaches—with a focus on *compositionality*—emerged from the study of formal programming language semantics but is becoming ever more important in Theoretical Computer Science.

References

1. Abramsky, S., Coecke, B.: A categorical semantics of quantum protocols. In: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS), 2004. pp. 415–425. IEEE (2004)

2. Backens, M.: The zx-calculus is complete for stabilizer quantum mechanics. *New Journal of Physics* **16**(9), 093021 (2014)
3. Baez, J., Erbele, J.: Categories in control. *Theory and Applications of Categories* **30**, 836–881 (2015)
4. Baez, J., Fong, B., Pollard, B.: A compositional framework for markov processes. *Journal of Mathematical Physics* **57**(033301) (2016)
5. Baez, J.C.: Network theory (2014), <http://math.ucr.edu/home/baez/networks/>, website (retrieved 15/04/2014)
6. Baez, J.C., Fong, B.: A compositional framework for passive linear networks. arXiv preprint arXiv:1504.05625 (2015)
7. Basold, H., Bonsangue, M.M., Hansen, H.H., Rutten, J.: (co)algebraic characterizations of signal flow graphs. In: *Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday. Lecture Notes in Computer Science*, vol. 8464, pp. 124–145. Springer (2014). https://doi.org/10.1007/978-3-319-06880-0_6
8. Bloom, S.L., Ésik, Z., Zsuzsa, B.: *Iteration theories: The equational logic of iterative processes*. Springer (1993)
9. Bonchi, F., Gadducci, F., Kissinger, A., Sobocinski, P., Zanasi, F.: Rewriting modulo symmetric monoidal structure. *CoRR* **abs/1602.06771** (2016)
10. Bonchi, F., Holland, J., Pavlovic, D., Sobociński, P.: Refinement for signal flow graphs. In: *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*. pp. 24:1–24:16 (2017). <https://doi.org/10.4230/LIPIcs.CONCUR.2017.24>, <https://doi.org/10.4230/LIPIcs.CONCUR.2017.24>
11. Bonchi, F., Holland, J., Piedeleu, R., Sobociński, P., Zanasi, F.: Diagrammatic algebra: from linear to concurrent systems. *Proceedings of the 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)* **3**, 1–28 (2019)
12. Bonchi, F., Pavlovic, D., Sobocinski, P.: Functorial semantics for relational theories. arXiv preprint arXiv:1711.08699 (2017)
13. Bonchi, F., Piedeleu, R., Sobociński, P., Zanasi, F.: Graphical affine algebra. In: *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. pp. 1–12 (2019)
14. Bonchi, F., Piedeleu, R., Sobocinski, P., Zanasi, F.: Contextual equivalence for signal flow graphs. In: *Goubault-Larrecq, J., König, B. (eds.) Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Proceedings. Lecture Notes in Computer Science*, vol. 12077, pp. 77–96. Springer (2020). https://doi.org/10.1007/978-3-030-45231-5_5
15. Bonchi, F., Sobociński, P., Zanasi, F.: A categorical semantics of signal flow graphs. In: *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR)*. pp. 435–450. Springer (2014)
16. Bonchi, F., Sobocinski, P., Zanasi, F.: Full abstraction for signal flow graphs. In: *Proceedings of the 42nd Annual ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*. pp. 515–526 (2015)
17. Bonchi, F., Sobocinski, P., Zanasi, F.: The calculus of signal flow diagrams I: linear relations on streams. *Information and Computation* **252**, 2–29 (2017)
18. Bonchi, F., Sobociński, P., Zanasi, F.: Interacting Hopf algebras. *Journal of Pure and Applied Algebra* **221**(1), 144–184 (2017)
19. Bruni, R., Gadducci, F.: Some algebraic laws for spans (and their connections with multi-relations). In: *RelMiS 2001*. Elsevier (2001)
20. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. *Theoretical Computer Science* **366**(1–2), 98–120 (2006)

21. Bruni, R., Melgratti, H., Montanari, U.: Connector algebras, petri nets, and bip. In: International Andrei Ershov Memorial Conference on Perspectives of System Informatics. pp. 19–38. Springer (2011)
22. Carboni, A., Walters, R.F.C.: Cartesian bicategories I. *Journal of Pure and Applied Algebra* **49**, 11–32 (1987)
23. Coecke, B., Duncan, R.: Interacting quantum observables. In: Proceedings of the 35th international colloquium on Automata, Languages and Programming (ICALP), Part II. pp. 298–310 (2008)
24. Coecke, B., Kissinger, A.: *Picturing Quantum Processes - A first course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press (2017)
25. Coya, B., Fong, B.: Corelations are the prop for extraspecial commutative frobenius monoids. *Theory and Applications of Categories* **32**(11), 380–395 (2017)
26. Coya, B.: *Circuits, Bond Graphs, and Signal-Flow Diagrams: A Categorical Perspective*. Ph.D. thesis, University of California Riverside (2018)
27. De Nicola, R., Hennessy, M.C.: Testing equivalences for processes. *Theoretical Computer Science* **34**(1-2), 83–133 (1984)
28. Dixon, L., Kissinger, A.: Open-graphs and monoidal theories. *Mathematical Structures in Computer Science* **23**(2), 308–359 (4 2013)
29. Fong, B.: *Causal Theories: A Categorical Perspective on Bayesian Networks*. Master’s thesis, Univ. of Oxford (2012), see arxiv.org/abs/1301.6201
30. Fong, B.: *The Algebra of Open and Interconnected Systems*. Ph.D. thesis, University of Oxford (2016)
31. Fong, B., Rapisarda, P., Sobociński, P.: A categorical approach to open and interconnected dynamical systems. In: LICS 2016 (2016)
32. Ghica, D.R., Jung, A.: Categorical semantics of digital circuits. In: Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design (FMCAD). pp. 41–48 (2016)
33. Ghica, D.R., Lopez, A.: A structural and nominal syntax for diagrams. In: Proceedings 14th International Conference on Quantum Physics and Logic (QPL). pp. 71–83 (2017)
34. Hasegawa, M.: Recursion from cyclic sharing: Traced monoidal categories and models of cyclic lambda calculi. pp. 196–213. Springer Verlag (1997)
35. Hoare, C.A.R.: *Communicating Sequential Processes*. Prentice Hall (1985)
36. Hyland, M., Power, J.: Symmetric monoidal sketches. In: Proceedings of the 2nd international ACM SIGPLAN conference on Principles and practice of declarative programming. pp. 280–288. ACM (2000). <https://doi.org/10.1145/351268.351299>
37. John C. Baez, Brandon Coya, F.R.: Props in network theory. *CoRR* **abs/1707.08321** (2017), <http://arxiv.org/abs/1707.08321>
38. Joyal, A., Street, R., Verity, D.: Traced monoidal categories. *Math Procs Cambridge Philosophical Society* **119**(3), 447–468 (4 1996)
39. Kelly, G.M., Laplaza, M.L.: Coherence for compact closed categories. *Journal of Pure and Applied Algebra* **19**, 193–213 (1980)
40. Kock, J.: *Frobenius algebras and 2D topological quantum field theories*. Cambridge University Press (2003)
41. Lack, S.: Composing PROPs. *Theory and Application of Categories* **13**(9), 147–163 (2004)
42. Lafont, Y.: Towards an algebraic theory of Boolean circuits. *Journal of Pure and Applied Algebra* **184**(2–3), 257–310 (2003)
43. Lawvere, W.F.: *Functorial Semantics of Algebraic Theories*. Ph.D. thesis (2004)
44. Mac Lane, S.: Categorical algebra. *Bulletin of the American Mathematical Society* **71**, 40–106 (1965)

45. Mason, S.J.: Feedback Theory: I. Some Properties of Signal Flow Graphs. MIT Research Laboratory of Electronics (1953)
46. Milius, S.: A sound and complete calculus for finite stream circuits. In: Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science (LICS). pp. 421–430 (2010)
47. Milner, R.: A Calculus of Communicating Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1982)
48. Mimram, S.: The structure of first-order causality. *Mathematical Structures in Computer Science* **21**, 65–110 (2 2011)
49. Pavlovic, D.: Monoidal computer I: Basic computability by string diagrams. *Information and Computation* **226**, 94–116 (2013)
50. Peterson, J.L.: Petri nets. *ACM Comput. Surv.* **9**(3), 223–252 (Sep 1977)
51. Piedeleu, R.: Picturing Resources in Concurrency. Ph.D. thesis, University of Oxford (2018)
52. Pirashvili, T.: On the PROP corresponding to bialgebras. *Cah. Top. Géom. Diff. Cat* **XLIII** (2002)
53. Rosebrugh, R., Sabadini, N., Walters, R.F.C.: Generic commutative separable algebras and cospans of graphs. *Theory and Application of Categories* **17**(6), 164–177 (2005)
54. Rutten, J.J.M.M.: A tutorial on coinductive stream calculus and signal flow graphs. *Theoretical Computer Science* **343**(3), 443–481 (2005)
55. Rutten, J.J.M.M.: Rational streams coalgebraically. *Logical Methods in Computer Science* **4**(3) (2008)
56. Sangiorgi, D., Walker, D.: *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA (2001)
57. Selinger, P.: A survey of graphical languages for monoidal categories. *Springer Lecture Notes in Physics* **13**(813), 289–355 (2011)
58. Shannon, C.E.: The theory and design of linear differential equation machines. Tech. rep., National Defence Research Council (1942)
59. Stefanescu, G.: *Network Algebra*. Discrete Mathematics and Theoretical Computer Science, Springer London (2000)
60. Stefanescu, G.: *Network Algebra*. Springer (2000)
61. Willems, J.C.: The behavioural approach to open and interconnected systems. *IEEE Control Systems Magazine* **27**, 46–99 (2007)
62. Willems, J.: Linear systems in discrete time. In: P.M.J. van den Hof, C.S., Heuberger, P. (eds.) *Festschrift on the occasion of the retirement of Okko Bosgra*. pp. 3–12 (2009)
63. Zanasi, F.: *Interacting Hopf Algebras: the theory of linear systems*. Ph.D. thesis, Ecole Normale Supérieure de Lyon (2015)
64. Zanasi, F.: The algebra of partial equivalence relations. *Electronic Notes in Theoretical Computer Science* **325**, 313–333 (2016)