# Full abstraction for digital circuits

DAN R. GHICA, University of Birmingham, United Kingdom
GEORGE KAYE, University of Birmingham, United Kingdom
DAVID SPRUNGER, University of Birmingham, United Kingdom

This paper refines the existing axiomatic semantics of digital circuits with delay and feedback, in which circuits are constructed as morphisms in a freely generated cartesian traced (dataflow) category. First, we give a cleaner presentation, making a clearer distinction between syntax and semantics, including a full formalisation of the semantics as *stream functions*. As part of this effort, we refocus the categorical framework through the lens of *string diagrams*, which not only makes reading equations more intuitive but removes bureaucracy such as associativity from proofs. We also extend the existing framework with a new axiom, inspired by the Kleene fixed-point theorem, which allows circuits with non-delay-guarded feedback, typically handled poorly by traditional methodologies, to be replaced with a series of finitely iterated circuits. This eliminates the possibility of infinitely unfolding a circuit; instead, one can always reduce a circuit to some (possibly undefined) value. To fully characterise the stream functions that correspond to digital circuits, we examine how the behaviour of the latter can be modelled using *Mealy machines*. By establishing that the translation between sequential circuits and Mealy machines preserves their behaviour, one can observe that circuits always implement monotone stream functions with finite stream derivatives.

CCS Concepts: • **Theory of computation → Axiomatic semantics**.

Additional Key Words and Phrases: digital circuits, traced monoidal categories, string diagrams, mealy machines, causal stream functions

## 1 INTRODUCTION

Sequential (digital) circuits are constructed by wiring together a set of basic components ('gates') which have clearly defined inputs and outputs. Gates operate on a discrete set of values, which are accepted as inputs and produced as outputs. Propagation of values along wires can be *delayed* using special componentry, the simplest of which is the D (from 'delay') *flip-flop*, which is controlled by a *clock*. The time-unit of the clock (the 'tick') is the time-unit of the delay.

Larger circuits can be constructed out of smaller circuits in three ways. They can be placed side-by-side, 'in parallel', in which case their respective inputs and outputs are also placed side-by-side. Alternatively, the outputs of one circuit can be fed into the inputs of the next circuit, 'in series', so that the resulting circuit has the inputs of the first circuit and the outputs of the second. Finally, some of the outputs of a circuit can be fed back as its own input. Using these three forms of composition, circuits of billions of elementary components can be created using a very small number of basic gates; circuits that control our computers, mobile phones, or car engines. And yet, surprisingly, the semantics of such circuits is not perfectly understood, in a sense which will be made clear immediately.

The three forms of composition and their inherent properties correspond to mathematical constructions called *symmetric traced monoidal categories* (STMCs). Moreover, sequential circuits can copy values (by forking a wire) and also discard them (by capping or stubbing a wire), which confers the additional categorical structure of a *Cartesian product*. STMCs equipped with Cartesian structure are called *dataflow categories*: this is a setting in which we can reason about sequential circuits *algebraically* (or *axiomatically*). By using the graph-based syntax of *string diagrams* [22] rather than a term-based syntax, we can give an effective method of *evaluating* circuits, i.e. an *operational semantics* [17, 18]. In this way it becomes possible to use semantic methodology which proved extremely successful in the study of programming languages in the context of sequential circuits.

Until recently, this line of theoretical development was not pursued because of a technical obstacle: the problem of *non-delay-guarded feedback*. In common engineering practice the application of feedback is not unrestricted but must always be 'guarded' by a delay. Although this restriction does not reduce the *expressiveness* of sequential circuits, it has negative practical and theoretical consequences. The practical inconvenience is that digital circuits cannot be treated as 'black boxes', because under the modelling assumptions whenever we feed back an output to an input of the same circuit a delay must exist somewhere along the wire. So we either add a possibly redundant delay just to be safe, or we must peek inside the box to see if a delay already exists. The theoretical drawback is that by restricting the application of the feedback we deny the STMC structure and, consequently, we invalidate the dataflow axioms, making the kind of desirable reasoning mentioned in the paragraph above impossible. Consequently, existing digital design tools simply reject such circuits with non-delay-guarded feedback, even though they can sometimes be useful [33]. But we do not want to overemphasise the usefulness of such circuits, which can be dismissed as hacks to be avoided by disciplined engineering. Our primary concern is having a more precise and fully compositional axiomatic framework for sequential digital circuits, and all the reasoning benefits such a framework entails.

The key idea for overcoming the technical obstacle represented by non-delay-guarded feedback is to move from a set of values to a *lattice*, those allowing the interpretation of both delay-guarded and non-delay-guarded feedback [32]. This idea, formulated in the context of a denotational semantics, was subsequently axiomatised categorically [17]. Using insights from string diagrams, which allow the derivation of an efficient graph-based representation of terms in the categorical term language, an operational semantics was then formulated [18], which is arguably more convenient to work with than the denotational or axiomatic models.

*Contribution.* In this paper we complete the categorical semantics by adding additional axioms to guarantee *full abstraction*: that is, an equivalence between the categorical and the denotational (stream) model of sequential circuits. Unlike theoretical models of programming languages, in which the challenge of full abstraction is on the denotational side, i.e. giving an interpretation of a language defined syntactically, in the case of sequential circuits the difficulties were in the opposite direction. The denotational model, or at least the desired denotational model, of digital circuits is an obvious one: streams and functions on streams. The challenge has been in finding the right syntax and the right axioms for describing sequential circuits *compositionally*. The first contribution of this paper is therefore to introduce additional axioms so that full abstraction can hold.

Moreover, not all stream functions correspond to digital circuits: some streams have an infinite number of stream derivatives, which contradicts the finite nature of circuits. The second contribution of this paper is to establish exactly which streams represent the behaviour of sequential circuits: to do this, we explore another interpretation of sequential circuits: that of Mealy machines [31]. It has been shown that a subset of stream functions are the carrier of the *final coalgebra* for Mealy machines [12], so we can exploit this connection to fully characterise the stream functions that correspond to digital circuits.

The contributions of this paper can be summed up as the dashed lines in Figure 1.

*Structure of the paper.* The remainder of the paper is structured as follows. In §2 the categorical framework of digital circuits will be recapped. In §3, this framework will be extended to handle circuits with arbitrary non-delay-guarded feedback. In §4 and §5 we will use Mealy machines to fully characterise the stream functions that correspond to digital circuits. Finally, §6 covers related and further work. The axioms of STMCs are listed in Appendix A.

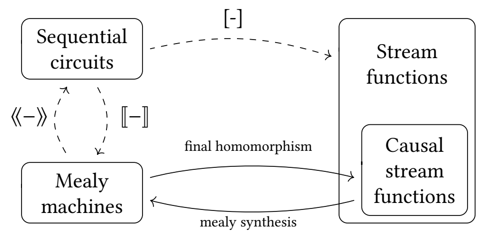



Fig. 1. Summary of the contributions of this paper.

## 2 DIGITAL CIRCUITS

This section will recap the categorical semantics of digital circuits, as defined in [17]. First the components of a circuits must be defined.

Definition 1 (Circuit signature). *Let $\Sigma$ be a tuple $(\mathcal{V}, \mathcal{G})$ where $\mathcal{V}$ is a finite set of* values *and $\mathcal{G}$ is a finite set of* gate symbols *$g$ with associated arities.*

A signature defines the *syntax* that can be used to build digital circuits.

Example 2. *Let $\mathcal{V}_\star$ be the set of values $\{\mathsf{t}, \mathsf{f}\}$, and let $\mathcal{G}_\star$ be the set $\{(AND, 2), (OR, 2), (NOT, 1)\}$. Throughout this paper, we will use the signature $\Sigma_\star = (\mathcal{V}_\star, \mathcal{G}_\star)$.*

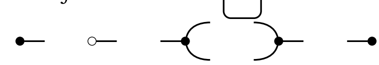
### 2.1 Combinational circuits

Circuit signatures are used to generate categories of circuits. Rather than reasoning purely with one-dimensional categorical terms, the graphical syntax of *string diagrams* [22, 39] can be used. In addition to being easier to read, this also adds the advantage that structural rules such as associativity are absorbed into the diagrams, thus greatly simplifying some proofs.

String diagrams work especially well with *props* [28], symmetric monoidal categories with natural numbers as objects and tensor product as addition. A string diagram box with $m$ input wires and $n$ output wires then corresponds very clearly to a morphism $m \to n$. In our diagrams we may coalesce wires together where appropriate: this should be clear from context.

Definition 3 (Combinational circuits). *For a signature $\Sigma = (\mathcal{V}, \mathcal{G})$, let $\mathbf{CCirc}_\Sigma$ be the symmetric strict monoidal prop generated freely by*

$$\boxed{v}\!- \;:\; 0 \to 1 \text{ for } v \in \mathcal{V} \qquad -\boxed{g}- \;:\; m \to 1 \text{ for } g \in \mathcal{G}$$

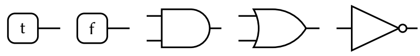


On the bottom line, the first two generators correspond to special values common to all circuits: a disconnected wire (no information) and a short circuit (too much information). The remaining generators are 'structural' generators for forking, joining and stubbing wires. Since the category is freely generated, circuit morphisms are defined by juxtaposing the generators in a given signature sequentially or in parallel with each other, the symmetry and the identity .

A circuit is called *closed* if it has no inputs, and *open* otherwise. To save space, circuit morphisms consisting of multiple generators may be drawn as single boxes. To distinguish these from individual rectangular gate boxes $-\boxed{g}-$, they are drawn as $-\boxed{F}-$. The word 'value' may refer to any of $\boxed{v}\!-$, $\bullet\!-$, or $\circ\!-$.

Definition 4. *A circuit is* passive *if it contains no values.*

Example 5. *The signature $\Sigma_\star$ defines the generators:*

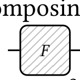


Constructing circuits in this manner is similar to [29], but with the addition of the •— and ○— generators. In the next sections the constructions will diverge as delay and feedback are added.

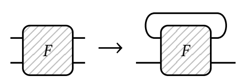
### 2.2 Circuits with feedback and delay

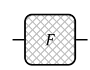
Most circuits are not merely formed of combinational logic: they have a notion of *delay*.

Definition 6 (Temporal circuits). *Let **TCirc**$_\Sigma$ be the category obtained by freely extending **CCirc**$_\Sigma$ with an additional generator -○- .*

The new generator -○- represents a delay of one tick, the smallest unit of time. Longer delays can be modelled by composing together multiple delay generators. Circuits that may contain values or delays are shaded:  .

Circuits can also contain *feedback*. This can be modelled by adding a *trace operator* [23]: connecting some of the outputs to the inputs.



If a circuit may contain delay or feedback, it is hatched:  .

Definition 7 (Sequential circuits). *Let **SCirc**$_\Sigma$ be the category obtained by freely extending **TCirc**$_\Sigma$ with a trace operator.*

This means that **SCirc**$_\Sigma$ is a *symmetric traced monoidal category*, or STMC. The axioms of STMCs are standard: they are listed in the appendix.

Remark 8. *A particularly important example of a category with a trace is the category of complete lattices with continuous functions between them. (This is a special case of complete pointed partial orders and continuous functions.) In this category, a trace is defined using least fixed points. Suppose $f : S \times M \to S \times N$ is a monotone function between lattices. For each $m \in M$, there is then a monotone endofunction on $S$ given by $s \mapsto \pi_S(f(s,m))$. Let $\mu_m$ be the least fixed point of this function. Then letting $Tr^S(f)(m) = \pi_N(f(\mu_m, m))$ defines a trace.*

### 2.3 Semantics

So far circuits have only been constructed *syntactically*: they have no computational behaviour associated with them yet. To add *semantics* to circuits, first the signature must be interpreted in some domain.

Definition 9 (Interpretation). *Let $\Sigma = (\mathcal{V}, \mathcal{G})$ be a circuit signature. A interpretation of $\Sigma$ is a tuple $\mathcal{I} = (\mathbf{V}, \mathcal{I}_\mathcal{V}, \mathcal{I}_\mathcal{G})$ where*

- ***V** is a finite lattice.*
- *$\mathcal{I}_\mathcal{V}$ is a bijective function $\mathcal{V} \to \mathbf{V} \setminus \{\bot, \top\}$.*
- *$\mathcal{I}_\mathcal{G}$ is a map from each gate symbol $(g, m) \in \mathcal{G}$ to a monotone function $\overline{g} : \mathbf{V}^m \to \mathbf{V}$ such that $\overline{g}(\bot^m) = \bot$.*

The special values •— and ○— correspond to the values $\bot$ and $\top$ in the lattice respectively.

| $\wedge$ | $\bot$ | f | t | $\top$ |
|---|---|---|---|---|
| $\bot$ | $\bot$ | f | $\bot$ | f |
| f | f | f | f | f |
| t | $\bot$ | f | t | $\top$ |
| $\top$ | f | f | $\top$ | $\top$ |

| $\vee$ | $\bot$ | f | t | $\top$ |
|---|---|---|---|---|
| $\bot$ | $\bot$ | $\bot$ | t | t |
| f | $\bot$ | f | t | $\top$ |
| t | t | t | t | t |
| $\top$ | t | $\top$ | t | $\top$ |

| $\neg$ | |
|---|---|
| $\bot$ | $\bot$ |
| t | f |
| f | t |
| $\top$ | $\top$ |

Fig. 2. Truth tables for the gates in $\Sigma_\star$ under $\mathcal{I}_\star$.

Example 10. *Recall the signature $\Sigma_\star = (\mathcal{V}_\star, \mathcal{G}_\star)$ from Example 2. The values can be interpreted in the four value lattice $\mathbf{V}_\star = \{\bot, \mathsf{t}, \mathsf{f}, \top\}$, where the join $\mathsf{t} \sqcup \mathsf{f} = \top$ and the meet $\mathsf{t} \sqcap \mathsf{f} = \bot$. Subsequently, we will interpret the gates using Belnap logic [3]: the truth tables are listed in Figure 2. Let $\mathcal{I}_\star = (\mathbf{V}_\star, \{\mathsf{t} \mapsto \mathsf{t}, \mathsf{f} \mapsto \mathsf{f}\}, \{AND \mapsto \wedge, OR \mapsto \vee, NOT \mapsto \neg\})$.*

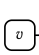
The lattice does not have to be as simple as $\mathbf{V}_\star$. For example, there could be two levels in the lattice, containing ‘weak’ and ‘strong’ versions of the values, which models the values used in metal-oxide-semiconductor field-effect transistors (MOSFET).

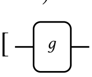
Semantics for digital circuits can be described in terms of *streams*, infinite sequences of values over time. In particular, circuits implement *stream functions.* The set of streams of $\mathbf{V}$ is denoted $\mathbf{V}^\omega$. For a stream $\sigma : \mathbf{V}^\omega$, its element at tick $k \in \mathbb{N}$ is written $\sigma(k)$. Similarly, for a stream function $f$ and some input stream $\sigma$, its behaviour at tick $k$ is written as $f(\sigma)(k)$. The unique element of $\mathbf{V}^0$ is written $(\bullet)$.

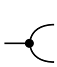
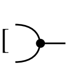
Definition 11. *Let $\mathbf{V}$ be a lattice. Then, let $\mathbf{Stream_V}$ be the traced prop with morphisms $m \to n$ the stream functions $(\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$.*

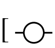
Since $\mathbf{Stream_V}$ is also a prop, semantics can be assigned to morphisms in $\mathbf{SCirc}_\Sigma$ by a *prop morphism*, a morphism in the category of props.

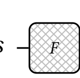
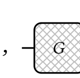
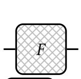
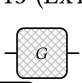
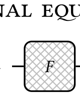
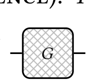
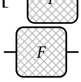
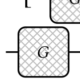
Definition 12. *Let $[-]_\mathcal{I} : \mathbf{SCirc}_\Sigma \to \mathbf{Stream_V}$ be a prop morphism. Since $\mathbf{SCirc}_\Sigma$ is freely generated, $[-]_\mathcal{I}$ is defined solely by its action on the generators.*

[  ]$_\mathcal{I}(\sigma)(0) = \mathcal{I}_\mathcal{V}(v)$ [  ]$_\mathcal{I}(\sigma)(k+1) = \bot$ [  ]$_\mathcal{I}(\sigma)(0) = \top$ [  ]$_\mathcal{I}(\sigma)(k+1) = \bot$

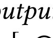
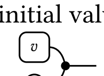
[  ]$_\mathcal{I}(\sigma)(k) = \bot$ [  ]$_\mathcal{I}(\sigma)(k) = \mathcal{I}_\mathcal{G}(g)(\sigma(k))$

[  ]$_\mathcal{I}(\sigma)(k) = (\sigma(k), \sigma(k))$ [  ]$_\mathcal{I}(\sigma)(k) = (\sigma(k) \sqcup \sigma(k))$ [  ]$_\mathcal{I}(\sigma)(k) = (\bullet)$

[  ]$_\mathcal{I}(\sigma)(0) = \bot$ [  ]$_\mathcal{I}(\sigma)(k+1) = \sigma(k)$

Definition 13 (Extensional equivalence). *For two sequential circuits* , , *if* [  ]$_\mathcal{I}$ = [  ]$_\mathcal{I}$ *then*  *and*  *are said to be* extensionally equivalent, *written*  $\approx_\mathcal{I}$ .

Definition 14. *Let $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ be the category obtained by quotienting $\mathbf{SCirc}_\Sigma$ by $\approx_\mathcal{I}$.*

Remark 15. *When viewed through the stream semantics, the delay generator*  *can be seen as a D flip-flop, where inputs are delayed for one tick before being output. One may wonder how the concept of an* initial value *stored in a flip-flop can be modelled, as* [  ]$_\mathcal{I}(\sigma)(0) = \bot$. *This can be constructed as*  *and denoted succinctly as* .

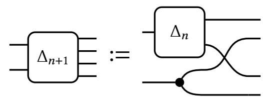
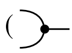
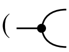


Fig. 3. Defining the diagonal $\Delta_{n+1}$.

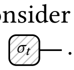
Quotienting by extensional equivalence gives $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ some structure. (, ) is a commutative monoid and (, ) is a cocommutative comonoid; together they form a *bialgebra*. This results in many 'theorems for free', such as the cocommutativity of the fork.

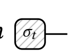
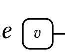
Theorem 16 ([17]). *$\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ is cartesian with $\Delta_n$ and $\diamond_n$.*

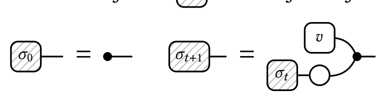
A monoidal category is *cartesian* if it has two natural families of maps $\Delta_n : n \to n+n$ (*diagonal*) for *copying* data and $\diamond_n : n \to 0$ (*discard*) for *deleting* data. The diagonal is constructed in $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ as $\Delta_0 =$  and $\Delta_{n+1}$ as shown Figure 3; the discard should be obvious.

$\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ being cartesian makes it a *traced cartesian* (or dataflow [13]) category. This will be explored further in §3.

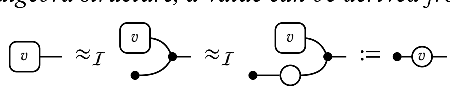
### 2.4 Axiomatisation

In the presence of delay, rather than thinking of the inputs to our circuits as single values, it is more useful to consider sequences of values over time, or *waveforms*. A waveform is drawn as a small shaded box .

Definition 17 (Waveform). *A $t$-waveform  is defined for a given head value  as*



Remark 18. *Using the bialgebra structure, a value can be derived from a waveform of length* 1.



*Therefore one can reason purely with waveforms.*

The cost of comparing the behaviour of circuits with an input waveform using extensional equivalence can be prohibitive.

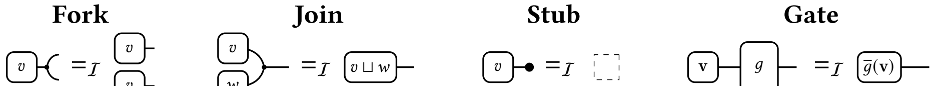
Proposition 19 ([19]). *Two sequential circuits containing no more than n delay generators are extensionally equivalent if and only if they produce the same outputs for all waveforms of length up to* $|\boldsymbol{V}|^n + 1$.

This establishes a superexponential upper bound for checking extensional equivalence.

While one could use extensional equivalence to reason with circuits, it would be far more advantageous to characterise $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ *axiomatically*. [17] presents eight such axioms that hold for any signature and interpretation.

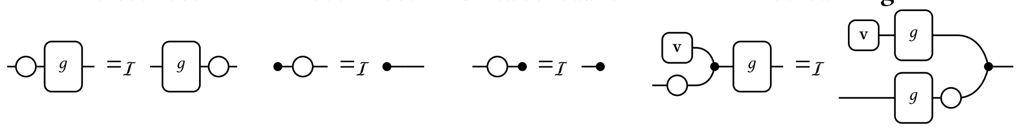
The first four, shown below, correspond to how a fork copies a value, a join coalesces two values, a stub discards a value, and a gate (monotonically) produces a value that depends entirely on its inputs.

**Fork** **Join** **Stub** **Gate**



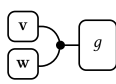
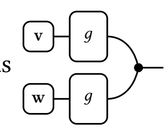
The last four axioms characterise delays:

**Timelessness** **Disconnect** **Unobservable** **Streaming**



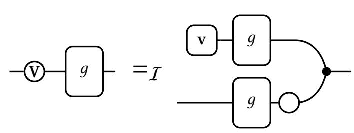
*Timelessness* implies that gates compute instantaneously so we can freely shift delays around them akin to retiming [30]; *disconnect* indicates that a delayed disconnect can be considered as disconnected already; and dually *unobservable delay* states that any delay that will be discarded can already be considered discarded.

The *streaming* axiom is perhaps the most unexpected. Intuitively, it says that the join operator is 'almost' a natural transformation. In general this is not the case:

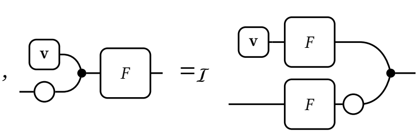
 is not the same as 

as the former reduces to $\boxed{\bar{g}(\mathrm{v} \sqcup \mathrm{w})}$ while the latter reduces to $\boxed{\bar{g}(\mathrm{v}) \sqcup \bar{g}(\mathrm{w})}$. However, when one of the inputs is guarded by a delay then there is no need to actually combine the inputs, so a guarded form of naturality holds.

Remark 20. *Observe that when using the 'flip-flop' notation from Remark 15, the streaming axiom can be expressed as*



For brevity, streaming can be generalised for arbitrary combinational circuits.

Lemma 21. *For any combinational circuit* $\boxed{F}$, 

Proof. By induction over the structure of $\boxed{F}$ and repeatedly applying *streaming*. □

For any axiomatisation to be a suitable candidate, it is essential that the axioms still preserve the *behaviour* of circuits.

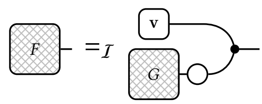
Theorem 22. $\boxed{F} =_{\mathcal{I}} \boxed{G} \Rightarrow \boxed{F} \approx_{\mathcal{I}} \boxed{G}$.

Proof. By applying $[-]_{\mathcal{I}}$ to the left and right hand side of each rule and asserting that the same stream function is obtained. □

## 3 PRODUCTIVITY

The goal of developing an axiomatisation is to obtain *full abstraction*: a correspondence between the axiomatic and the denotational semantics. In the case of sequential circuits, this means if some inputs are provided to a circuit, it can always be reduced to some value: the output of its corresponding function in the stream semantics. This is known as *productivity*.

Definition 23 (Productivity). *Given a closed sequential circuit* $F$, *it is called* productive *if there exist values* $\mathrm{v}$ *and sequential circuit* $G$ *such that*



*All* combinational and temporal circuits are productive.

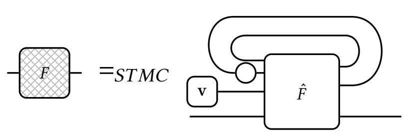
Theorem 24 (Extensionality [17]). *For a $t$-waveform* $\sigma_t$ *and a combinational circuit* $F$, *there exists a $t$-waveform* $\tau_t$ *such that* $\sigma_t\, F =_{\mathcal{I}} \tau_t$. *Moreover, for a temporal circuit* $F$ *there exists a $t'$-waveform* $\tau_{t'}$ *such that* $\sigma_t\, F =_{\mathcal{I}} \tau_{t'}$.

However, there is no guarantee that this is the case for sequential circuits under the current axiomatisation.

### 3.1 Handling sequential circuits

In [18], an efficient strategy for obtaining a sequence of values over time for certain sequential circuits is presented. This will be recapped here.

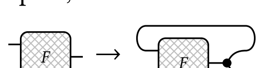
Lemma 25 (Global trace-delay form). *For any sequential circuit* $F$, *there exists a passive combinational* $\hat{F}$ *and values* $\mathrm{v}$ *such that*



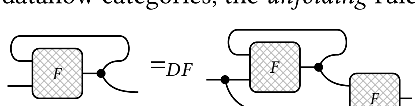
*by axioms of STMCs.*

Proof. Any trace can become a 'global trace' by applying *tightening* and *superposing*. For the delays, we can use *yanking* to create a feedback loop, and then follow the same procedure, using *sliding* to shift delays to the correct place. □

To reason further, the fact that $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ is a *dataflow* category must be used to *unfold* the traced circuit. To do this, the notion of a a *Conway operator* must be introduced. The action of a Conway operator on a circuit is to copy its outputs, feed back one of them and output the other.

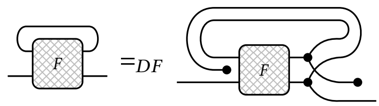


By applying the axioms of dataflow categories, the *unfolding* rule can be derived:



Theorem 26 ([20]). *A cartesian category is traced if and only if it has a family of Conway operators.*

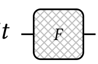
The construction of a Conway operator from a trace using the axioms of dataflow categories is as follows:

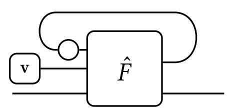


By applying this construction to the global trace-delay form of a circuit, and then unfolding, axioms can be used in the new copy of the circuit.

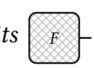
### 3.2 Delay-guarded feedback

Even with unfolding, not all circuits can be reduced to a waveform in this way.

Definition 27 (Delay-guarded feedback). *A sequential circuit*  *has* delay-guarded feedback *if its global trace-delay form is*

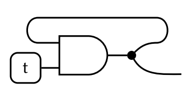


*i.e. all feedback passes through a delay.*

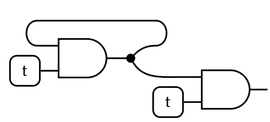
Theorem 28 (Productivity [18]). *All closed sequential circuits*  *with delay-guarded feedback are productive.*

When considering circuits with non-delay-guarded feedback, this guarantee does not necessarily hold.

Example 29. *Consider the circuit*

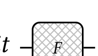
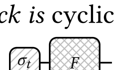


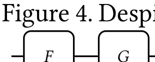
*In the stream semantics, the circuit produces the constant stream $\bot :: \bot :: \cdots$, but this cannot be obtained axiomatically. The only option is to unfold, which results in the following circuit:*



*Once again, the only option is to unfold: this circuit will never reduce to a value.*

One might ask if the delay-guarded feedback condition should be enforced in order to assert that all circuits are productive. However, not all circuits with non-delay-guarded feedback are unproductive.

Definition 30 (Cyclic combinational circuits). *A sequential circuit*  *with non-delay-guarded feedback is* cyclic combinational *if for any $t$-waveform* *, there exists a $t'$-waveform*  *such that*  $=_{\mathcal{I}}$ .

There are instances where cyclic combinational feedback is acceptable and even beneficial [33]. One such example [32] can be seen in Figure 4. Despite the presence of feedback, there is a sequence of axioms that reduces the circuit to  or  depending on the signal $c$

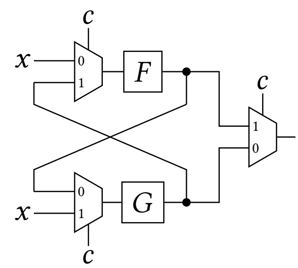



Fig. 4. An example of a cyclic combinational circuit, adapted from [32, Fig. 1], where $F$ and $G$ are arbitrary combinational circuits.

given to the multiplexers. The feedback is merely a way of sharing resources between the two branches of the circuit.

Moreover, enforcing that all feedback is delay-guarded would also prevent the construction of a freely generated traced monoidal category. There do exist weakenings of traced categories in which this 'delay-guardedness' principle holds, in the form of categories with *feedback* [24] or *delayed trace* [41]. However, these are not suitable in this context as the unfolding rule would not hold.

### 3.3 Eliminating non-delay-guarded feedback

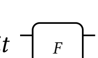
When interpreted in the stream semantics, Example 29 *does* produce a stream of values: a constant stream of the 'undefined' $\perp$ value. To achieve full abstraction, an axiomatic characterisation must be found that demonstrates this behaviour. Inspiration can be gleaned from the following theorems:

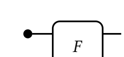
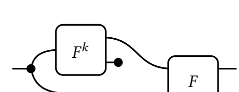
Theorem 31 (Kleene fixed-point theorem [42]). *Let $\mathbf{V}$ be a finite lattice, and let $f : \mathbf{V} \to \mathbf{V}$ be a Scott-continuous function. Then $f$ has a least fixed point in $\mathbf{V}$: the supremum of $\{f^n(\perp) \mid n \in \mathbb{N}\}$.*

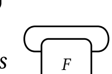
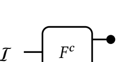
Lemma 32. *For a monotonic function $f : \mathbf{V}^{n+m} \to \mathbf{V}^n$, let $f_i : \mathbf{V}^m \to \mathbf{V}^n$ be defined for $i \in \mathbb{N}$ as $f_0(x) = f(\perp, x)$, $f_{k+1}(x) = f(f_k(x), x)$. Let $c$ be the length of the longest chain in the value lattice $\mathbf{V}^n$. Then, for $j > c$, $f_c(x) = f_j(x)$.*

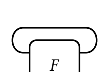
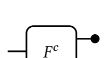
Proof. Since $f$ is monotonic, it has a least fixed point by the Kleene fixed-point theorem. This will either be some value $v$ or, since $\mathbf{V}$ is finite, the $\top$ element. The most iterations of $f$ it would take to obtain this fixpoint is $c$, i.e. the function produces a value one step up the lattice each time. □

Monotone functions are the semantics of combinational circuits, so this suggests a new family of axioms.

Definition 33 (Instant feedback). *For a combinational circuit*  $: x + m \to x + n$ *and $i \in \mathbb{N}$, let $F^i : m \to x + n$ be defined as*

$F^0 :=$  $F^{k+1} :=$ 

*Let $c$ be the length of the longest chain in $\mathbf{V}$. Then, the instant feedback rule is*  $=_{\mathcal{I}}$  .

Lemma 34.  $\approx_{\mathcal{I}}$  .

Proof. By Lemma 32. □

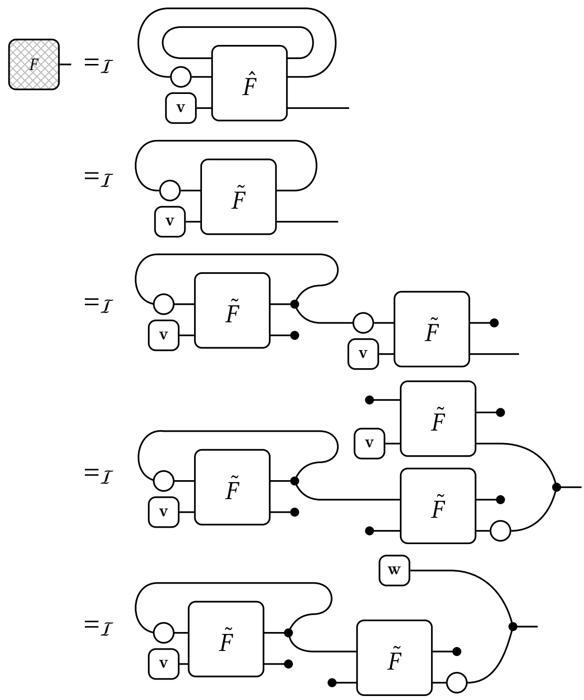
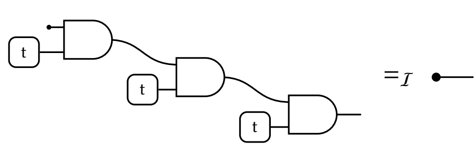


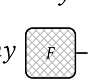
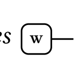
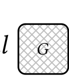

Fig. 5. Proof of Theorem 36.

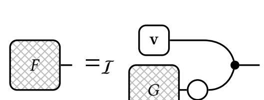
Example 35. *Recall the circuit in Example 29. By applying the instant feedback rule, the circuit is manipulated into a form from which it* can *be reduced to a value:*




*As desired, the circuit collapses to the undefined*  *value.*

This rule can be used to refine Theorem 28. Rather than restricting to the subset of circuits with delay-guarded feedback, it can be shown that *any* closed circuit can be reduced to a waveform.

Theorem 36 (Productivity II). *Given any*  *, there exist values*  *and sequential*  *such that*




Proof. Shown in Fig. 5. First the circuit is put into global trace-delay form. Then *instant feedback* is applied to create another combinational circuit. Next we unfold. By using the fact that  is the unit of the join, streaming can be applied. Since the 'top' copy is a closed combinational circuit, it can be reduced to a tensor of values by *extensionality*. Thus the circuit has an instantaneous component and a delayed component. □

One might ask how it is possible to recover a value for each tick of the clock in the presence of unguarded feedback. Observe that the value obtained may not necessarily be a 'useful' value, such as t or f: it may also be the 'undefined' value $\bot$ as in Example 35. This can be thought of as a default value for unbounded recursion: although the computation will never terminate, the use of a finite lattice enforces that a fixpoint is always reached, so the undefined $\bot$ value can be returned.

By repeatedly applying productivity, a stream of values can be obtained for *any* sequential circuit $-\boxed{F}-$, given some inputs $\boxed{\mathrm{v}}-$. Since the axioms preserve input-output behaviour, the elements of this stream are the same elements that would be found by applying these inputs to the stream $[\![ -\boxed{F}- ]\!]_{\mathcal{I}}$. Therefore this selection of axioms is enough to obtain *full abstraction* for digital circuits.

## 4 MEALY MACHINES

Although all sequential circuits correspond to stream functions, not all stream functions correspond to digital circuits. Since circuits are composed of a finite number of gates, they cannot specify an infinite number of behaviours. This means that any stream function which performs a completely different behaviour for each tick cannot model a circuit.

To fully characterise the stream functions that have corresponding digital circuits, it is useful to first interpret digital circuits as *Mealy machines* [31], a kind of finite state automaton. The reasoning behind this is that a subset of stream functions form the *final coalgebra* for Mealy machines [37]. This coalgebraic structure can be exploited to characterise the stream function semantics.

Moreover, it is well known that classical digital circuits can be constructed from binary Mealy machines [27]: thus it is also a good litmus test to see if the same applies when lifted to lattices and applied to categorical circuit morphisms.

Definition 37 (Mealy machine [31]). *For a finite set $\mathbf{M}$ and a possibly infinite set $\mathbf{N}$, a (finite)* Mealy machine *with interface $(\mathbf{M}, \mathbf{N})$ is a tuple $(\mathbf{S}, s_0, T, O)$ where*

- *$\mathbf{S}$ is a finite set of* register states.
- *$s_0 \in \mathbf{S}$ is an* initial state.
- *$T : \mathbf{S} \to \mathbf{S}^{\mathbf{M}}$ is a* transition function.
- *$O : \mathbf{S} \to \mathbf{N}^{\mathbf{M}}$ is an* output function.

$\mathbf{M}$ is the *input space* and $\mathbf{N}$ is the *output space*. $\mathbf{S}$ is the *state space*, containing all of the possible states the internal registers in the system can take.

Definition 38 (Bisimulation [12]). *For two $(\mathbf{M},\mathbf{N})$-Mealy machines $A = (\mathbf{S}, s_0, T, O)$ and $B = (\mathbf{S}', s'_0, T', O')$, a* bisimulation *is a relation $R \subseteq \mathbf{S} \times \mathbf{S}'$ such that for all $(s, s') \in \mathbf{S} \times \mathbf{S}'$ and $a \in \mathbf{M}$,*

$$O(s)(a) = O'(s')(a) \qquad (T(s)(a), T'(s')(a)) \in R$$

*The machines $A$ and $B$ are* bisimilar*, written $A \equiv B$, if there exists a bisimulation $R$ between them, and $(s_0, s'_0) \in R$.*

If two Mealy machines are bisimilar then they are *observationally equivalent* [34]: given their respective initial states both machines will produce the same outputs.

Definition 39. *For a Mealy machine $A$, let $!A$ be defined as its* output stream*, defined for an input stream $\boldsymbol{x}_0 :: \boldsymbol{x}_1 :: \boldsymbol{x}_2 :: \cdots$ as*

$$O(s_0)(\boldsymbol{x}_0) :: O(T(s_0)(\boldsymbol{x}_0))(\boldsymbol{x}_1) :: O(T(T(s_0)(\boldsymbol{x}_0))(\boldsymbol{x}_1))(\boldsymbol{x}_2) :: \cdots$$

Corollary 40 ([12]). *Mealy machines are bisimilar if and only if their output streams are equal.*

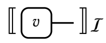
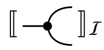
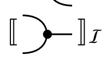
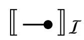
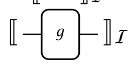
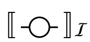
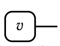
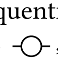
| | | | | |
|---|---|---|---|---|
|  | $0 \xrightarrow{1} 1$ | $T(r_0)(\bullet) = (\bot)$ | $O(r_0)(\bullet) = (r_0)$ | $s_0 = (v)$ |
|  | $1 \xrightarrow{0} 2$ | $T(\bullet)(x_0) = (\bullet)$ | $O(\bullet)(x_0) = (x_0, x_0)$ | $s_0 = (\bullet)$ |
|  | $2 \xrightarrow{0} 1$ | $T(\bullet)(x_0, x_1) = (\bullet)$ | $O(\bullet)(x_0, x_1) = (x_0 \sqcup x_1)$ | $s_0 = (\bullet)$ |
|  | $1 \xrightarrow{0} 0$ | $T(\bullet)(x_0) = (\bullet)$ | $O(\bullet)(x_0) = (\bullet)$ | $s_0 = (\bullet)$ |
|  | $m \xrightarrow{0} 1$ | $T(\bullet)(\mathbf{x}^m) = (\bullet)$ | $O(\bullet)(\mathbf{x}^m) = \overline{g}(\mathbf{x}^m)$ | $s_0 = (\bullet)$ |
|  | $1 \xrightarrow{1} 1$ | $T(r_0)(x_0) = (x_0)$ | $O(r_0)(x_0) = (r_0)$ | $s_0 = (\bot)$ |

Table 1. The action of $[\![-]\!]_{\mathcal{I}}$ on generators in $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$.

### 4.1 The prop of $\mathcal{I}$-Mealy machines

When considering a sequential circuit morphism, there are two components that determine state: values  and delays , each of which 'contain' a value in $\mathbf{V}$. Therefore, a possible state space for a circuit with $r$ values and delays will be $\mathbf{V}^r$. Since the outputs and next state are determined purely from the current state and the inputs, a corresponding Mealy machine can be defined. For now, we will consider a class of Mealy machines where the transition and output are determined using functions from the interpretation $\mathcal{I}$.

Definition 41. *Let $\mathcal{I} = (\mathbf{V}, \mathcal{I}_{\mathcal{V}}, \mathcal{I}_{\mathcal{G}})$ be an interpretation. Then let $\mathbf{Func}_{\mathcal{I}}$ be the (non-traced) prop freely generated over the functions $\mathbf{V}^m \to \mathbf{V}^n$ in $\mathcal{I}_{\mathcal{G}}$, with composition and tensor defined as usual on functions.*

Definition 42 ($\mathcal{I}$-Mealy machine). *For an interpretation $\mathcal{I}$ and $m, n, s \in \mathbb{N}$, a $\mathcal{I}$-Mealy machine is a Mealy machine $(\mathbf{V}^s, s_0, T, O)$ with interface $(\mathbf{V}^m, \mathbf{V}^n)$ such that $T$ and $O$ are morphisms in $\mathbf{Func}_{\mathcal{I}}$.*

For a $\mathcal{I}$-Mealy machine where $\mathbf{M} = \mathbf{V}^m$, $\mathbf{N} = \mathbf{V}^n$ and $\mathbf{S} = \mathbf{V}^s$, we write it as $m \xrightarrow{s} n$ for short. For two tuples $\mathbf{v} \in \mathbf{V}^m$ and $\mathbf{w} \in \mathbf{V}^n$, let their *concatenation* $\mathbf{v} \curlyvee \mathbf{w} \in \mathbf{V}^{m+n}$ be the tuple containing the elements of $\mathbf{v}$ followed by the elements of $\mathbf{w}$.

Definition 43. *Let $\mathbf{Mealy}_{\mathcal{I}}$ be the traced cartesian prop, where the morphisms $m \to n$ are the $\mathcal{I}$-Mealy machines $m \xrightarrow{s} n$ for some $s \in \mathbb{N}$, with the operations defined below.*

The composition of $A : m \xrightarrow{s} n$ and $B : n \xrightarrow{s'} p$ is the *cascade product* $C : m \xrightarrow{s+s'} p$, where

$$T_C(\mathbf{s}_A \curlyvee \mathbf{s}_B)(\mathbf{x}) = T_A(\mathbf{s}_A)(\mathbf{x}) \curlyvee T_B(\mathbf{s}_B)(O(\mathbf{s}_A)(\mathbf{x}))$$
$$O_C(\mathbf{s}_A \curlyvee \mathbf{s}_B)(\mathbf{x}) = O_B(\mathbf{s}_B)(O_A(\mathbf{s}_A)(\mathbf{x}))$$

The tensor of $A : m \xrightarrow{s} n$ and $B : p \xrightarrow{s'} q$ is the *direct product* $C : m + p \xrightarrow{s+s'} n + q$ where

$$T_C(\mathbf{s}_A \curlyvee \mathbf{s}_B)(\mathbf{x}) = T_A(\mathbf{s}_A)(\mathbf{x}) \curlyvee T_B(\mathbf{s}_B)(\mathbf{x}_B)$$
$$O_C(\mathbf{s}_A \curlyvee \mathbf{s}_B)(\mathbf{x}) = O_A(\mathbf{s}_A)(\mathbf{x}) \curlyvee O_B(\mathbf{s}_B)(\mathbf{x}_B)$$

The copy and discard machines are *stateless*: they only have an output component.

$$T_{\Delta}(\bullet)(\mathbf{x}) = (\bullet) \quad O_{\Delta}(\bullet)(\mathbf{x}) = \mathbf{x} + \mathbf{x}$$
$$T_{\diamond}(\bullet)(\mathbf{x}) = (\bullet) \quad O_{\diamond}(\bullet)(\mathbf{x}) = (\bullet)$$

For a Mealy machine $A : n + m \xrightarrow{s} n$, an *iteration* $\Phi$ is defined over $\mathbb{N}$ as $\Phi(0)(\mathbf{s})(\mathbf{x}) = \bot^n$ and $\Phi(k+1)(\mathbf{s})(\mathbf{x}) = O_A(\mathbf{s})(\Phi(k)(\mathbf{s})(\mathbf{x}) \curlyvee \mathbf{x})$. The *fixpoint* of $A$ is computed as $\mathrm{fix}(\mathbf{s})(\mathbf{x}) =$

$\bigsqcup_{k\in\mathbb{N}} \Phi(k)(\mathbf{s})(\mathbf{x})$. The effect of a Conway operator on $A$ is then $B : m \xrightarrow{s} n$, where

$$T_B(\mathbf{s})(\mathbf{x}) = T_A(\mathbf{s})(\mathsf{fix}(\mathbf{s})(\mathbf{x}) \vee \mathbf{x}) \quad O_B(\mathbf{s})(\mathbf{x}) = \mathsf{fix}(\mathbf{s})(\mathbf{x}) \vee \mathbf{x}$$

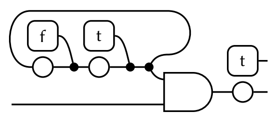
The trace can then be defined using the Conway operator.

### 4.2 Mealy machines from circuits

$\mathcal{I}$-Mealy machines can then be constructed from sequential circuits by using a prop morphism.

Definition 44. *Let* $\llbracket - \rrbracket : \mathbf{SCirc}_{\Sigma,\mathcal{I}} \to \mathbf{Mealy}_{\mathcal{I}}$ *be a traced prop morphism with its action on generators defined in Table 1.*

Example 45. *Consider the circuit*




*This contains three values and three delays, so the state space for a corresponding Mealy machine will be* $\mathbf{V}^6$. *One possible initial state is* $(\mathsf{f}, \bot, \mathsf{t}, \bot, \mathsf{t}, \bot)$. *The functions for this Mealy machine would be* $T(r_0, \cdots, r_5)(x) = (\bot, r_2 \sqcup r_3, \bot, r_0 \sqcup r_1, \bot, (r_2 \sqcup r_3) \wedge x)$, $O(r_0, \cdots, r_5)(x) = (r_4, r_5)$.

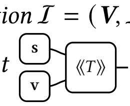
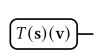
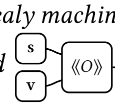
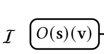
When translating between circuits and Mealy machines, it is essential that their behaviour is preserved: they must still be interpreted as the same stream function. This ensures that reasoning performed with Mealy machines also applies to circuits, and vice versa.

Theorem 46. *For any sequential circuit* $F$, $[F]_{\mathcal{I}} = !(\llbracket F \rrbracket_{\mathcal{I}})$.

Proof. By induction on the structure of $F$. The generators, composition and tensor are trivial; for trace the proof is by translating the circuit into the corresponding Conway operator and reasoning that if the inductive hypothesis holds, then applying a Conway operator to it corresponds to finding the fixpoint of equal functions. □

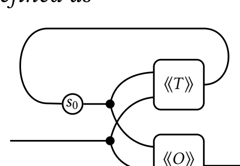
### 4.3 Circuits from Mealy machines

A prop morphism can also be used to construct a circuit from some $\mathcal{I}$-Mealy machine.

Lemma 47. *For an interpretation* $\mathcal{I} = (\mathbf{V}, \mathcal{I}_{\mathcal{V}}, \mathcal{I}_{\mathcal{G}})$ *and* $\mathcal{I}$*-Mealy machine, there exists combinational circuits* $\langle\!\langle T \rangle\!\rangle$ *and* $\langle\!\langle O \rangle\!\rangle$ *such that*  $\approx_{\mathcal{I}}$  *and*  $\approx_{\mathcal{I}}$ .

Proof. $T$ and $O$ are morphisms in $\mathbf{Func}_{\mathcal{I}}$, so they are compositions of functions in $\mathcal{I}_{\mathcal{G}}$. Therefore, they can be expressed as some combination of gate generators. □

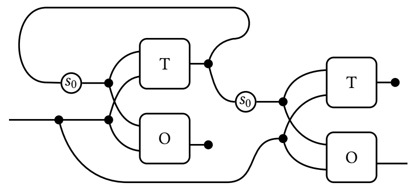
Definition 48. *Let* $\langle\!\langle - \rangle\!\rangle_{\mathcal{I}} : \mathbf{Mealy}_{\mathcal{I}} \to \mathbf{SCirc}_{\Sigma,\mathcal{I}}$ *be a prop morphism, with its effect on a* $\mathcal{I}$*-Mealy machine* $A = (\mathbf{V}^s, s_0, T, O)$ *defined as*



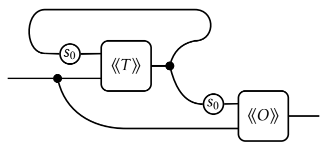
Now it must be checked that $\langle\!\langle - \rangle\!\rangle_{\mathcal{I}}$ preserves behaviour.

Theorem 49. *For any Mealy machine* $A$, $!A = [\langle\!\langle A \rangle\!\rangle_{\mathcal{I}}]_{\mathcal{I}}$.

Proof. By unfolding $[\langle\!\langle A \rangle\!\rangle_{\mathcal{I}}]_{\mathcal{I}}$ it can be seen that the stream semantics are equal to the output stream.




After applying axioms of dataflow categories the following circuit is obtained:

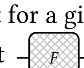
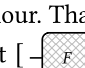



Since $\langle\!\langle T \rangle\!\rangle$ and $\langle\!\langle O \rangle\!\rangle$ are defined to have the behaviour of $T$ and $O$, the behaviour of $[\langle\!\langle A \rangle\!\rangle_{\mathcal{I}}]_{\mathcal{I}}(\sigma)(0)$ will be $O(s_0)(\sigma(0))$. By unfolding repeatedly each tick of $!A$ can be obtained. $\square$

The results of Theorems 46 and 49 show that one can freely translate between a sequential circuit and an $\mathcal{I}$-Mealy machine. This is crucial, as it means the rich coalgebraic structure of the latter can be used to also show results for circuits.

So far, only Mealy machines that have 'nice' structure have been considered, in that the contents of the states is visible and the transition and output functions are morphisms in $\mathbf{Func}_{\mathcal{I}}$. However, Mealy machines are often specified in terms of 'black-box' states with some transition table. In the next section, it will be also shown how certain classes of these *arbitrary* Mealy machines can be translated into circuits in a behaviour-preserving manner.

## 5 STREAMS

Given a circuit, its behaviour as a stream function can be produced by using the prop morphism $[-]_{\mathcal{I}}$. Now we consider the converse problem: synthesising a circuit for a given behaviour. That is, given a function $f : (\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$, is there a recipe for a circuit  such that [  $]_{\mathcal{I}} = f$? As we have mentioned previously, this is not possible for general functions $f$, but this section will detail the necessary conditions for this to be possible, and outline how this circuit synthesis can be accomplished. With this construction, we will be able to identify the class of stream functions that make up the image of $[-]_{\mathcal{I}}$.

### 5.1 Synthesising a Mealy machine

As noted in the previous section, a Mealy machine consumes and produces streams. A fact from coalgebra [37] is that a subset of stream functions $f : \mathbf{M}^\omega \to \mathrm{N}^\omega$, known as *causal* stream functions, can themselves be organised into a family of (infinite) Mealy machines.

Definition 50 (Causal stream function). *A stream function $f : \boldsymbol{M}^\omega \to \boldsymbol{N}^\omega$ is* causal *if for all $i \in \mathbb{N}$ and $\sigma, \tau \in \boldsymbol{M}^\omega$, it holds that $f(\sigma)(i) = f(\tau)(i)$ whenever $\sigma(j) = \tau(j)$ for all $0 \leq j \leq i$.*

Basically, this means the $i$th output of a causal stream function can only depend on the first $i$ inputs. Before giving these functions a Mealy machine structure, we recall two operations on streams. A stream $\mathbf{V}^\omega$ is equipped with two operations: the *initial value* $\mathsf{i}(-) : \mathbf{V}^\omega \to \mathbf{V}$, which produces the 'head' of the stream; and *stream derivative* $\mathsf{d}(-) : \mathbf{V}^\omega \to \mathbf{V}^\omega$, a function producing the 'tail' of the stream, defined as $\mathsf{d}(\sigma)(i) = \sigma(i+1)$. For the $n$th stream derivative, we write $\mathsf{d}^n$.

These operations can be extended to act on causal stream functions:

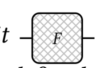
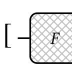
Definition 51 (Functional stream derivative [37]). *Suppose $f : \boldsymbol{M}^\omega \to \boldsymbol{N}^\omega$ is a causal stream function and let $a \in \boldsymbol{M}$. The* initial output of $f$ on input $a$ *is* $f[a] = \mathrm{i}(f(a :: \sigma)) \in \boldsymbol{N}$ *for any $\sigma \in \boldsymbol{M}^\omega$. The* functional stream derivative of $f$ on input $a$ *is a function $f_a : \boldsymbol{M}^\omega \to \boldsymbol{N}^\omega$ given by $f_a(\sigma) = \mathrm{d}(f(a :: \sigma))$.*

The causality of $f$ ensures $f[a]$ does not depend on the choice of $\sigma$. $f_a$ can be thought of as acting as $f$ would 'had it seen $a$ first'.

When the initial output is set as the output function, and the functional stream derivative is set as the transition function, a causal stream function $f : \mathbf{M}^\omega \to \mathbf{N}^\omega$ is a nearly an ($\mathbf{M}$,$\mathbf{N}$)-Mealy machine as described in Definition 37, with two caveats: there is no designated start state and the state space is infinite. However, given a particular function $f$, the minimal Mealy machine that has $f$ as the output stream can be found.

Proposition 52 ([37]). *If $f : \boldsymbol{M}^\omega \to \boldsymbol{N}^\omega$ is a causal stream function, let $S$ be the least set of causal stream functions that includes $f$ and is closed under functional stream derivatives: i.e. for all $h \in S$ and $a \in \boldsymbol{M}$, $h_a \in S$. Then the Mealy machine $\mathrm{S_f} = (S, f, T_S, O_S)$ where $T_S(h)(a) = h_a$ and $O_S(h)(a) = h[a]$, has the smallest state space of Mealy machines with the property that $!\mathrm{S_f} = f$.*

So in order to synthesise a finite circuit, the stream function $f$ must have finitely many stream derivatives.

Example 53. *Recall the circuit*  *from Example 45. Let $f : \boldsymbol{V}^\omega \to \boldsymbol{V}^\omega = [$*  *$]_{\mathcal{I}}$. The corresponding stream function is defined as*

$$f(\sigma)(i) = \begin{cases} (\mathrm{t}, \bot) & \textit{if } i = 0 \\ (\bot, \sigma(2k)) & \textit{if } i = 2k+1 \\ (\bot, \mathrm{f}) & \textit{if } i = 2k+2 \end{cases}$$

*From the first state $s_0 = f$, the output is $f[v] = (\mathrm{t}, v)$. The stream $s_1 = f_\bot$ is defined as $(s_1)(\sigma) = (f(\bot :: \sigma)) = (\bot, \mathrm{f}), (\bot, \sigma(0)), \cdots$. $s_2 = f_{\mathrm{t}}$, $s_3 = f_{\mathrm{f}}$ and $s_4 = f_\top$ are defined similarly but are distinct states. The output of states $1-4$ is always $(\bot, \mathrm{f})$ and the transition is always to the same new state $s_5$. For all $v''$, $s_5[v''] = (\bot, v'')$. Now consider $(s_5)_\bot$, defined as $(s_5)(\sigma) = d^3(f(\bot :: v :: \bot :: \sigma)) = (\bot, \mathrm{f}) :: (\bot, \sigma(0)) :: \cdots$: this is the same as $s_1$. Similarly, inputting $\mathrm{t}$, $\mathrm{f}$ and $\top$ produces $s_2$, $s_3$ and $s_4$ respectively. So we have fully specified the Mealy machine: it is illustrated in Figure 6.*

The state set of the derived Mealy machine is not necessarily a power of $\mathbf{V}$, so this is not yet an $\mathcal{I}$-Mealy machine. This means it cannot be translated directly into a sequential circuit and a correspondence established.

## 5.2 Synthesising a monotone Mealy machine

To obtain an $\mathcal{I}$-Mealy machine, each state in $\mathbf{S_f}$ must be *encoded* as a power of $\mathbf{V}$. This is a standard procedure for constructing a digital circuit from a Mealy machine [27]. However, in this context this encoding must also allow the definition of monotone transition and output functions.

Definition 54 (Monotone stream function). *Let $\boldsymbol{M}$, $\boldsymbol{N}$ be partially ordered sets. Then, a causal stream function $f : \boldsymbol{M}^\omega \to \boldsymbol{N}^\omega$ is monotone if, for any $a \in \boldsymbol{M}$, $f[a]$ is monotone, and $f_a$ is a monotone stream function.*

For the transition function to be monotone, the stream functions themselves must be comparable in some way. This is quite a challenge since the space of causal stream functions does not have an ordering on it! However, causal stream functions *between partially ordered sets* have a natural ordering.

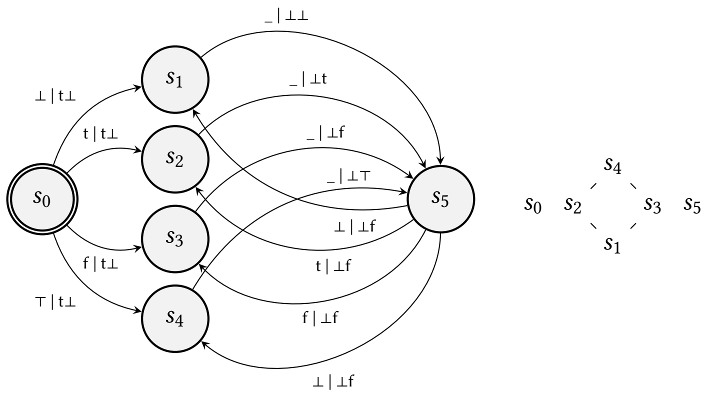



Fig. 6. On the left, the Mealy machine from Example 53, where a transition label $v \mid w$ indicates an input $v$ and an output $w$. On the right, the corresponding state ordering.

Definition 55. *Let $\mathbf{M}$ be a set and let $\mathbf{N}$ be a partially ordered set. Then there is an induced order on $\mathbf{N}^\omega$ by comparing each element pointwise. Let $f, g : \mathbf{M}^\omega \to \mathbf{N}^\omega$ be two causal stream functions. We say $f \preceq g$ if $f(\sigma) \leq_{\mathbf{N}^\omega} g(\sigma)$ for all $\sigma \in \mathbf{M}^\omega$.*

Lemma 56. *The relation $\preceq$ is a partial order.*

For a Mealy machine $\mathsf{S}_\mathbf{f}$ derived as in the previous section, the elements of its state set $S$ are causal stream functions $(\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$, so they inherit this ordering.

Example 57. *The state ordering for the Mealy machine derived in Example 53 is shown in Figure 6.*

To obtain a $\mathcal{I}$-Mealy machine, the output and transition functions on the derived Mealy machine must be monotone with respect to this ordering. We first show that these functions are monotone in their state component.

Proposition 58. *Let $f, g : \mathbf{M}^\omega \to \mathbf{N}^\omega$ be monotone stream functions such that $f \preceq g$. Then $f[a] \leq_\mathbf{N} g[a]$ and $f_a \preceq g_a$ for all $a \in \mathbf{M}$.*

Proof. Let $\sigma \in \mathbf{M}^\omega$ be arbitrary. By Definition 51 and the fact that $\mathsf{i}(-)$ is monotone, $f[a] = \mathsf{i}(f(a :: \sigma)) \leq_\mathbf{N} \mathsf{i}(g(a :: \sigma)) = g[a]$. Similarly, $f_a(\sigma) = \mathsf{d}(f(a :: \sigma)) \leq_{\mathbf{N}^\omega} \mathsf{d}(g(a :: \sigma)) = g_a(\sigma)$ by definitions and the monotonicity of stream derivative, so $f_a \preceq g_a$. □

Now we can check that the functions obtained in the Mealy machine construction are monotone.

Proposition 59. *Suppose $\mathbf{M}$ and $\mathbf{N}$ are partially ordered sets, let $f : \mathbf{M}^\omega \to \mathbf{N}^\omega$ be a monotone causal function, and let $\mathsf{S}_\mathbf{f}$ be a Mealy machine defined as in Proposition 52. Then, the functions $O_S$ and $T_S$ are monotone.*

Proof. Proposition 58 shows these functions are monotone for fixed input letters: it remains to show that the functions are monotone for fixed functions from $S$. Let $h \in S$ and suppose $a \leq_\mathbf{M} a'$. Since $h$ is monotone, $O_S(h)(a) = h[a] = \mathsf{i}(h(a :: \sigma)) \leq_\mathbf{N} \mathsf{i}(h(a' :: \sigma)) = O_S(f)(a')$, and similarly for $T_S$. As these functions are monotone in both components, they are monotone overall. □

This means that the ordering is suitable for constructing an $\mathcal{I}$-Mealy machine: by using it, tuples can be assigned to each state that respect the monotonicity of the transition and output.

Definition 60 (State assignment). *Let $S$ be a state space $S$ equipped with an ordering $\preceq$, and let $r = |S|$. Then, the state assignment $\gamma : S \to \mathbf{V}^r$ is defined as*

$$\gamma(s_i)(j) = \begin{cases} \top & \text{if } s_j \preceq s_i \\ \bot & \text{otherwise} \end{cases}$$

Lemma 61. *In the context of Definition 60, if $s \preceq s'$, then $\gamma(s) \sqsubseteq \gamma(s')$.*

### 5.3 Synthesising an $\mathcal{I}$-Mealy machine

With these conditions, it is now possible to characterise the stream functions that correspond to $\mathcal{I}$-Mealy machines, and subsequently to sequential circuits. There is one final point to consider: to reconstruct an $\mathcal{I}$-Mealy machine from a Mealy machine $\mathrm{S_f}$, the transition and output functions must be expressible as morphisms in $\mathbf{Func}_{\mathcal{I}}$.

Definition 62 (Functional completeness). *An interpretation $\mathcal{I} = (\mathbf{V}, \mathcal{I}_{\mathcal{V}}, \mathcal{I}_{\mathcal{G}})$, is called* functionally complete *if all monotone functions $\mathbf{V}^m \to \mathbf{V}^n$ are morphisms in $\mathbf{Func}_{\mathcal{I}}$.*

Example 63. *The set of Belnap logic functions $\{\wedge, \vee, \neg\}$ from Example 10 is functionally complete.*

Proposition 64. *Let $\mathcal{I}$ be functionally complete, and for a Mealy machine $\mathrm{S_f} = (S, s_0, T, O)$ derived from an $\mathcal{I}$-circuit function as above, let $r = |S|$. Then, there exist morphisms in $\mathbf{Func}_{\mathcal{I}}$ $T' : \mathbf{V}^r \to \mathbf{V}^{r(\mathbf{V}^m)}$ and $O' : \mathbf{V}^r \to \mathbf{V}^{n(\mathbf{V}^m)}$ such that for any $s \in S$ and $a \in \mathbf{V}^m$, if $T(s)(a) = s'$, then $T'(\gamma(s))(a) = \gamma(s')$, and if $O(s)(a) = v'$, then $O'(\gamma(s))(a) = v'$.*

Proof. $\mathcal{I}$ is functionally complete. □

To formally define the exact set of stream functions that can be translated into $\mathcal{I}$-Mealy machines, we first observe that causal functions of streams are in one-to-one correspondence with special sequences of functions called approximants.

Definition 65. *Let $f : \mathbf{M}^\omega \to \mathbf{N}^\omega$ be a causal function. Its* approximants *are the functions $(f_i : \mathbf{M}^{i+1} \to \mathbf{N})_{i \in \mathbb{N}}$ given by $f_i(m_0, m_1, \cdots, m_i) = f(m_0 :: m_1 :: \cdots :: m_i :: \sigma)(i)$ for any $\sigma \in \mathbf{M}^\omega$.*

Conversely, given any sequence of functions with these domains and ranges, one can construct a causal function. Using the notion of approximants, the conditions on a stream function that corresponds to an $\mathcal{I}$-Mealy machine can be defined.

Definition 66 ($\mathcal{I}$-circuit function). *Let $\mathcal{I} = (\mathbf{V}, \mathcal{I}_{\mathcal{V}}, \mathcal{I}_{\mathcal{G}})$ be a functionally complete interpretation. A causal stream function $f : (\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$ is called an $\mathcal{I}$-circuit function if*

- *for each tuple $w \in \mathbf{V}^k$, there exists a family of tuples $w' \in \mathbf{V}^p$ and $w'' \in \mathbf{V}^r$, $r > 0$ such that, for any $i \in \mathbb{N}$, $f_{k+p}(w \vee w') = f_{k+p+ri}(w \vee w' \vee w'')$; and*
- *each approximant $f_i$ belongs to $\mathbf{Func}_{\mathcal{I}}$*

In the above definition, one can think of the (possibly empty) tuple $w' \in \mathbf{V}^p$ acting as the word to transition out of some *prefix* behaviour and the tuple $w''$ as the word that transitions through some *periodic* behaviour.

Lemma 67. *$\mathcal{I}$-circuit functions (1) have finitely many stream derivatives; and (2) are monotone.*

Proof. (1) follows from the first condition in the definition. (2) follows since each approximant is monotone. This means the initial output is monotone; the initial output of the first stream derivative is monotone, and so on. □

Proposition 68. *For any $\mathcal{I}$-Mealy machine $A$, $!A$ is an $\mathcal{I}$-circuit function.*

Proof. For the first condition, let $w$ be the word that transitions from the initial state of $A$ to some state $s$. Then, let the word $w'$ be the (possibly empty) word that transitions from $s$ to a state $s'$ such that there exists a word $w''$ that transitions from $s'$ to $s'$. There always must be such a state $s'$ as the state set is finite and the transition function is deterministic: at some point a state must reoccur. The second condition holds since each approximant of $!A$ is constructed by composing $T$ and $O$, which are defined to belong to $\mathbf{Func}_{\mathcal{I}}$. □

Definition 69. *Given a functionally complete interpretation $\mathcal{I} = (\mathbf{V}, \mathcal{I}_{\mathcal{V}}, \mathcal{I}_{\mathcal{G}})$, let $\boldsymbol{CStream}_{\mathcal{I}}$ be the subcategory of $\boldsymbol{Stream}_{\mathbf{V}}$ with only the $\mathcal{I}$-circuit functions as morphisms.*

The translation between $\mathcal{I}$-circuit streams and $\mathcal{I}$-Mealy machines can now be formalised as a prop morphism.

Definition 70. *Let $\mathcal{I}$ be functionally complete. Given a $\mathcal{I}$-circuit function $f$, let $\mathrm{S}_{\mathbf{f}} = (S, f, T, O)$ be the Mealy machine obtained as in Proposition 52; let $\gamma$ be its state assignment as in Definition 60, and let $\gamma^{\star}(S)$ be defined as its pointwise application to elements in $S$. Let $T'$ and $O'$ be functions defined as in Proposition 64. Finally, let $\langle - \rangle_{\mathcal{I}} : \boldsymbol{CStream}_{\mathcal{I}} \to \boldsymbol{Mealy}_{\mathcal{I}}$ be a prop morphism with its behaviour defined as $\langle f \rangle_{\mathcal{I}} = (\gamma^{\star}(S), \gamma(f), T', O')$.*

Theorem 71. *Let $\mathcal{I}$ be functionally complete. Then, for any $\mathcal{I}$-Mealy machine $A$ and $\mathcal{I}$-circuit function $f$, $\langle !A \rangle_{\mathcal{I}} \equiv A$ and $!\langle f \rangle_{\mathcal{I}} = f$.*

Proof. For the first statement, $!A$ is defined to be the output stream of $A$. When $\langle !A \rangle$ is constructed, $O(\gamma(!A))(\sigma(0))$ is defined to be $!A(\sigma)(0) = O(s_0)(\sigma(0))$, so the initial states of $\langle !A \rangle$ and $A$ are bisimilar. The transition function on input $a$ is defined as $\mathrm{d}(f(a :: \sigma)) = f(a :: \sigma)(1) :: f(a :: \sigma)(2) :: \cdots . f(a :: \sigma) = O(T(s_0)(a))(\sigma(0))$, so the next states are also bisimilar, and so on.

For the second statement, $\langle f \rangle_{\mathcal{I}}$ is defined such that for stream $\sigma$, $O(\gamma(f))(\sigma(0)) = f(\sigma)(0)$, $O(T(\gamma(f))(\sigma(0)))(\sigma(1)) = f(\sigma)(1)$, and so on. Therefore the output stream will be the original stream function $f$. □

Therefore, the translation between $\mathcal{I}$-Mealy machine and $\mathcal{I}$-circuit functions is behaviour-preserving: this can be used to show the final result.

Theorem 72. *Let $\mathcal{I}$ be functionally complete. Then, $\boldsymbol{SCirc}_{\Sigma,\mathcal{I}} \cong \boldsymbol{CStream}_{\mathcal{I}}$.*

Proof. Let $-\boxed{F}-$ be a sequential circuit. First we show that $-\boxed{F}- \approx_{\mathcal{I}} \langle\!\langle \langle [ -\boxed{F}- ]_{\mathcal{I}} \rangle_{\mathcal{I}} \rangle\!\rangle_{\mathcal{I}}$. By Thm. 71, the translation from stream functions to Mealy machines is behaviour-preserving, so $!\langle [ -\boxed{F}- ]_{\mathcal{I}} \rangle_{\mathcal{I}} = [ -\boxed{F}- ]_{\mathcal{I}}$. Then by Thm. 49, the translation from Mealy machines to sequential circuits is behaviour-preserving, so $[ \langle\!\langle \langle [ -\boxed{F}- ]_{\mathcal{I}} \rangle_{\mathcal{I}} \rangle\!\rangle_{\mathcal{I}} ]_{\mathcal{I}} = [ -\boxed{F}- ]_{\mathcal{I}}$. By definition of extensional equivalence, $\langle\!\langle \langle [ -\boxed{F}- ]_{\mathcal{I}} \rangle_{\mathcal{I}} \rangle\!\rangle_{\mathcal{I}} \approx_{\mathcal{I}} -\boxed{F}-$.

Now we show $f = [ \langle\!\langle \langle f \rangle_{\mathcal{I}} \rangle\!\rangle_{\mathcal{I}} ]_{\mathcal{I}}$. By Thm. 71, $!\langle f \rangle_{\mathcal{I}} = f$, so by Thm. 46, $[ \langle\!\langle \langle f \rangle_{\mathcal{I}} \rangle\!\rangle_{\mathcal{I}} ]_{\mathcal{I}} = f$. □

Since there is an isomorphism between $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ and $\mathbf{CStream}_{\mathcal{I}}$, the latter fully characterises the stream functions that correspond to digital circuits.

## 6 CONCLUSION, RELATED AND FURTHER WORK

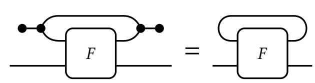
We have refined a framework for constructing sequential circuits syntactically as morphisms in a symmetric traced monoidal category. These circuits can be given semantics as *stream functions*: however, it is easier to reason with axioms that can reduce a circuit into a simpler one, with the aim of obtaining a sequence of values over time. To enable this strategy to be used with circuits with *non-delay-guarded feedback*, we proposed a new axiom compatible with the stream semantics, which allows instant feedback to be represented as a finite iteration of circuits.

To support this work, we have formally characterised the stream functions that correspond to digital circuits as causal stream functions with finite derivatives, such that the transition and output functions are monotone.

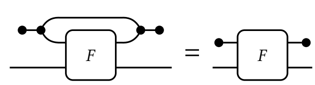
### 6.1 Related work

While the use of string diagrams as a graphical syntax for (traced) monoidal categories has existed for some time [22, 23], there has recently been an explosion in their use for practical applications such as quantum protocols [1], signal flow diagrams [9, 10], linear algebra [8, 11, 43], electronic circuits [4] and dynamical systems [2, 16].

While these diagrammatic frameworks all use compositional circuits in some way, the nature of digital circuits mean there are some differences in our system. In many of the applications listed, the join and the fork form a *Frobenius structure*: this induces a compact closed structure on the category and enforces that wires are bidirectional. This means that a trace can be constructed by using the join and the fork:




Conversely, since wires in digital circuits must be left-right oriented at their endpoints, the join and a fork form a *bialgebra.* This equips our category with a *traced* structure rather than a compact closed one; subsequently, attempting to define trace in the same way as above fails:




Using a trace is not the only way of enforcing this unidirectionality of wires while retaining loops. *Categories with feedback* were first introduced in [24] as a weakening of traced monoidal categories that removes the yanking axiom, effectively enforcing that *all* traces are delay-guarded. In [15] Mealy machines are characterised as a category with feedback, since each transition is synchronous. This makes sense when compared with our context, as all 'instant' feedback is eliminated by using the fixpoint operator, leaving all remaining trace 'delay-guarded'. *Categories with delayed trace* [41] weaken the notion further, by removing the dinaturality axiom: this prevents the unfolding rule from being derivable.

The link between Mealy machines and digital circuits is well known [31] and is used extensively in circuit design. Conversely, the links between the former and causal stream functions using coalgebras is a more recent development [35–38]. Using the coalgebraic framework, [12] presents an alternative method of specifying Mealy machines using *Mealy logic formulas*: this serves as a contrast to our more compositional approach.

### 6.2 Future work

While the categorical circuit framework is indeed advantageous for reasoning about digital circuits, it does present an 'idealised' perspective. Issues such as fan-out, the number of gates that can be controlled by a single signal, is not modelled: a signal can be forked arbitrarily many times at no cost. Similarly, components such as *amplifiers*, which can non-deterministically increase an undefined $\perp$ signal to a t or f might also be an interesting addition to the framework.

Another line of work, on the more practical side, is that of automating the reasoning framework. There may be many rewrites that can be applied to a given circuit, so working by hand could quickly become tedious. Instead, it would be easier to implement the reduction strategy specified in [18] to allow for *automated diagrammatic reasoning*, in a vein similar to that of the proof assistants *Quantomatic* [26], *homotopy.io* [14] and *Cartographer* [40].

Reasoning with string diagrams may be simple by hand, but it is not an efficient syntax to work with computationally. For an efficient operational semantics, string diagrams must be translated to combinatorial graphs with concrete sets of edges and vertices. This was touched on informally in [18], which used *framed point graphs*. Contrastingly, recent work in string diagram rewriting [5–7] have used *hypergraphs* in order perform rewriting modulo Frobenius structure. There has been work to adapt this framework for dataflow categories [25], however it would be interesting to see if this be expanded to perform rewriting modulo *bialgebraic* structure, as can be found in digital circuits.

## REFERENCES


[1] Samson Abramsky and Bob Coecke. 2004. A Categorical Semantics of Quantum Protocols. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.* 415–425. https://doi.org/10.1109/LICS.2004.1319636

[2] John C. Baez and Jason Erbele. 2015. Categories in Control. *arXiv:1405.6881 [quant-ph]* (May 2015). arXiv:1405.6881 [quant-ph]

[3] Nuel D. Belnap. 1977. A Useful Four-Valued Logic. In *Modern Uses of Multiple-Valued Logic*, J. Michael Dunn and George Epstein (Eds.). Springer Netherlands, Dordrecht, 5–37. https://doi.org/10.1007/978-94-010-1161-7_2

[4] Guillaume Boisseau and Paweł Sobociński. 2021. String Diagrammatic Electrical Circuit Theory. (June 2021). arXiv:2106.07763

[5] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobociński, and Fabio Zanasi. 2020. String Diagram Rewrite Theory I: Rewriting with Frobenius Structure. *CoRR* abs/2012.01847 (Dec. 2020). arXiv:2012.01847

[6] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobociński, and Fabio Zanasi. 2021. String Diagram Rewrite Theory II: Rewriting with Symmetric Monoidal Structure. *CoRR* abs/2104.14686 (April 2021). arXiv:2104.14686

[7] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. 2021. String Diagram Rewrite Theory III: Confluence with and without Frobenius. *CoRR* abs/2109.06049 (Sept. 2021). arXiv:2109.06049

[8] Filippo Bonchi, Robin Piedeleu, Pawel Sobociński, and Fabio Zanasi. 2019. Graphical Affine Algebra. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–12. https://doi.org/10.1109/LICS.2019.8785877

[9] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. 2014. A Categorical Semantics of Signal Flow Graphs. In *CONCUR 2014 – Concurrency Theory (Lecture Notes in Computer Science)*, Paolo Baldan and Daniele Gorla (Eds.). Springer, Berlin, Heidelberg, 435–450. https://doi.org/10.1007/978-3-662-44584-6_30

[10] Filippo Bonchi, Pawel Sobociński, and Fabio Zanasi. 2015. Full Abstraction for Signal Flow Graphs. *ACM SIGPLAN Notices* 50, 1 (Jan. 2015), 515–526. https://doi.org/10.1145/2775051.2676993

[11] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. 2017. Interacting Hopf Algebras. *Journal of Pure and Applied Algebra* 221, 1 (Jan. 2017), 144–184. https://doi.org/10.1016/j.jpaa.2016.06.002

[12] M. M. Bonsangue, Jan Rutten, and Alexandra Silva. 2008. Coalgebraic Logic and Synthesis of Mealy Machines. In *Foundations of Software Science and Computational Structures (Lecture Notes in Computer Science)*, Roberto Amadio (Ed.). Springer, Berlin, Heidelberg, 231–245. https://doi.org/10.1007/978-3-540-78499-9_17

[13] Virgil Emil Căzănescu and Gheorghe Ştefănescu. 1990. Towards a New Algebraic Foundation of Flowchart Scheme Theory. *Fundamenta Informaticae* 13, 2 (Jan. 1990), 171–210. https://doi.org/10.3233/FI-1990-13204

[14] Nathan Corbyn, Lukas Heidemann, Nick Hu, Calin Tataru, and Jamie Vicary. 2022. The Proof Assistant Homotopy.Io. https://doi.org/10.5281/zenodo.2540764

[15] Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Paweł Sobociński. 2021. A Canonical Algebra of Open Transition Systems. In *Formal Aspects of Component Software (Lecture Notes in Computer Science)*, Gwen Salaün and Anton Wijs (Eds.). Springer International Publishing, Cham, 63–81. https://doi.org/10.1007/978-3-030-90636-8_4

[16] Brendan Fong, Paweł Sobociński, and Paolo Rapisarda. 2016. A Categorical Approach to Open and Interconnected Dynamical Systems. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '16)*. Association for Computing Machinery, New York, NY, USA, 495–504. https://doi.org/10.1145/2933575.2934556

[17] Dan R. Ghica and Achim Jung. 2016. Categorical Semantics of Digital Circuits. In *2016 Formal Methods in Computer-Aided Design (FMCAD)*. 41–48. https://doi.org/10.1109/FMCAD.2016.7886659

[18] Dan R. Ghica, Achim Jung, and Aliaume Lopez. 2017. Diagrammatic Semantics for Digital Circuits. In *26th EACSL Annual Conference on Computer Science Logic (CSL 2017) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 82)*, Valentin Goranko and Mads Dam (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 24:1–24:16. https://doi.org/10.4230/LIPIcs.CSL.2017.24

[19] Dan R. Ghica, Achim Jung, and Aliaume Lopez. 2017. Diagrammatic Semantics for Digital Circuits (Technical Report). *CoRR* abs/1703.10247 (March 2017). arXiv:1703.10247

[20] Masahito Hasegawa. 1997. Recursion from Cyclic Sharing: Traced Monoidal Categories and Models of Cyclic Lambda Calculi. In *Typed Lambda Calculi and Applications (Lecture Notes in Computer Science)*, Philippe de Groote and J. Roger Hindley (Eds.). Springer, Berlin, Heidelberg, 196–213. https://doi.org/10.1007/3-540-62688-3_37

[21] Masahito Hasegawa. 2009. On Traced Monoidal Closed Categories. *Mathematical Structures in Computer Science* 19, 2 (April 2009), 217–244. https://doi.org/10.1017/S0960129508007184

[22] André Joyal and Ross Street. 1991. The Geometry of Tensor Calculus, I. *Advances in Mathematics* 88, 1 (1991), 55–112. https://doi.org/10.1016/0001-8708(91)90003-P

[23] André Joyal, Ross Street, and Dominic Verity. 1996. Traced Monoidal Categories. *Mathematical Proceedings of the Cambridge Philosophical Society* 119, 3 (April 1996), 447–468. https://doi.org/10.1017/S0305004100074338

[24] P. Katis, Nicoletta Sabadini, and Robert F. C. Walters. 2002. Feedback, Trace and Fixed-Point Semantics. *RAIRO - Theoretical Informatics and Applications* 36, 2 (April 2002), 181–194. https://doi.org/10.1051/ita:2002009

[25] George Kaye. 2021. Rewriting Graphically with Symmetric Traced Monoidal Categories. (March 2021). arXiv:2010.06319

[26] Aleks Kissinger and Vladimir Zamdzhiev. 2015. Quantomatic: A Proof Assistant for Diagrammatic Reasoning. In *Automated Deduction - CADE-25 (Lecture Notes in Computer Science)*, Amy P. Felty and Aart Middeldorp (Eds.). Springer International Publishing, Cham, 326–336. https://doi.org/10.1007/978-3-319-21401-6_22

[27] Zvi Kohavi and Niraj K. Jha. 2009. *Switching and Finite Automata Theory.* Cambridge University Press.

[28] Stephen Lack. 2004. Composing PROPs. *Theory and Applications of Categories* 13, 9 (2004), 147–163.

[29] Yves Lafont. 2003. Towards an Algebraic Theory of Boolean Circuits. *Journal of Pure and Applied Algebra* 184, 2 (Nov. 2003), 257–310. https://doi.org/10.1016/S0022-4049(03)00069-0

[30] Charles E. Leiserson and James B. Saxe. 1991. Retiming Synchronous Circuitry. *Algorithmica* 6, 1 (June 1991), 5–35. https://doi.org/10.1007/BF01759032

[31] George H. Mealy. 1955. A Method for Synthesizing Sequential Circuits. *The Bell System Technical Journal* 34, 5 (Sept. 1955), 1045–1079. https://doi.org/10.1002/j.1538-7305.1955.tb03788.x

[32] Michael Mendler, Thomas R. Shiple, and Gérard Berry. 2012. Constructive Boolean Circuits and the Exactness of Timed Ternary Simulation. *Formal methods in system design : an international journal* 40, 3 (2012), 283–329. https://doi.org/10.1007/s10703-012-0144-6

[33] Marc D. Riedel. 2004. *Cyclic Combinational Circuits.* Ph. D. Dissertation. California Institute of Technology, United States – California.

[34] J. J. M. M. Rutten. 2000. Universal Coalgebra: A Theory of Systems. *Theoretical Computer Science* 249, 1 (Oct. 2000), 3–80. https://doi.org/10.1016/S0304-3975(00)00056-6

[35] J. J. M. M. Rutten. 2005. Algebra, Bitstreams, and Circuits. *Software Engineering [SEN]* R 0502 (Jan. 2005).

[36] J. J. M. M. Rutten. 2005. A Coinductive Calculus of Streams. *Mathematical Structures in Computer Science* 15, 1 (Feb. 2005), 93–147. https://doi.org/10.1017/S0960129504004517

[37] J. J. M. M. Rutten. 2006. Algebraic Specification and Coalgebraic Synthesis of Mealy Automata. *Electronic Notes in Theoretical Computer Science* 160 (Aug. 2006), 305–319. https://doi.org/10.1016/j.entcs.2006.05.030

[38] J. J. M. M. Rutten. 2008. Rational Streams Coalgebraically. *Logical Methods in Computer Science* 4, 3 (Sept. 2008), 9. https://doi.org/10.2168/LMCS-4(3:9)2008 arXiv:0807.4073

[39] Peter Selinger. 2011. A Survey of Graphical Languages for Monoidal Categories. In *New Structures for Physics*, Bob Coecke (Ed.). Springer, Berlin, Heidelberg, 289–355. https://doi.org/10.1007/978-3-642-12821-9_4

[40] Paweł Sobociński, Paul Wilson, and Fabio Zanasi. 2019. Cartographer: A Tool for String Diagrammatic Reasoning. http://dx.doi.org/10.4230/LIPIcs.CALCO.2019.20. , 20:1-20:7 pages.

[41] David Sprunger and Shin-ya Katsumata. 2019. Differentiable Causal Computations via Delayed Trace. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS).* 1–12. https://doi.org/10.1109/LICS.2019.8785670

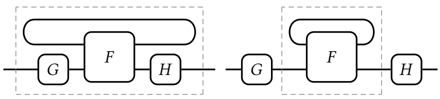
[42] V. Stoltenberg-Hansen, I. Lindström, and E. R. Griffor. 1994. *Mathematical Theory of Domains.* Cambridge University Press.

[43] Fabio Zanasi. 2015. *Interacting Hopf Algebras: The Theory of Linear Systems.* Ph. D. Dissertation. University of Lyon. arXiv:1805.03032

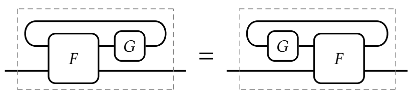

## A AXIOMS OF SYMMETRIC TRACED MONOIDAL CATEGORIES

These axioms were originally presented in [23].

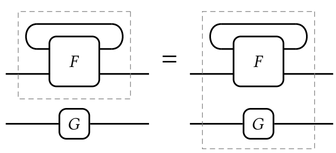
### A.1 Tightening (naturality)



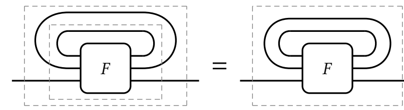

### A.2 Sliding (dinaturality)




### A.3 Superposing




### A.4 Vanishing




In [23], an additional vanishing axiom was presented; it is shown in [21] that this was in fact redundant.

### A.5 Yanking