A compositional theory of digital circuits

Dan R. Ghica, George Kaye, and David Sprunger

School of Computer Science, University of Birmingham

{d.r.ghica, g.j.kaye, d.sprunger}@cs.bham.ac.uk

Abstract

A theory is compositional if complex components can be constructed out of simpler ones on the basis of their interfaces, without inspecting their internals. Digital circuits, despite being studied for nearly a century and used at scale for about half that time, have until recently evaded a fully compositional theoretical understanding. The sticking point has been the need to avoid feedback loops that bypass memory elements, the so called 'combinational feedback' problem. This requires examining the internal structure of a circuit, defeating compositionality. Recent work remedied this theoretical shortcoming by showing how digital circuits can be presented compositionally as morphisms in a freely generated Cartesian traced (or dataflow) category. The focus was to support a better syntactical understanding of digital circuits, culminating in the formulation of novel operational semantics for digital circuits. In this paper we shift the focus onto the denotational theory of such circuits, interpreting them as functions on streams with to certain properties. These ensure that the model is fully abstract, i.e. the equational theory and the semantic model are in perfect agreement. To support this result we introduce two key equations: the first can reduce circuits with combinational feedback to circuits without combinational feedback via finite unfoldings of the loop, and the second can translate between open circuits with the same behaviour syntactically by reducing the problem to checking a finite number of closed circuits. The most important consequence of this new semantics is that we can now give a recipe that ensures a circuit always produces observable output, thus using the denotational model to inform and improve the operational semantics.

1 Introduction

Walther Bothe was awarded the 1954 Nobel Prize in physics for creating the electronic AND gate in 1924. Subsequently, exponential improvements in digital technology have led to the creation of the defining technologies of the modern world. It may therefore seem far-fetched that there are still theoretical gaps in mathematical and logical theories of digital circuits. And still, until recently, a fully *compositional* theory of digital circuits was not yet formulated.

By 'fully compositional' we mean that a larger circuit can be constructed out of smaller circuits and interconnecting wires without paying heed to the internal structure of these smaller circuits. If we try to do that we run into an obstacle: electrical connections can be created that inadvertently connect the output of some elementary gate back to its input such that no memory elements are encountered along the path. Such a path, called a 'combinational feedback loop' (or 'cycle'), causes the established mathematical theories of digital circuits to fail. Therefore, conventional digital design and engineering reject such circuits. To enforce this restriction, we need to always look inside circuits as we compose them, ensuring, each time a larger circuit is constructed, that no illegal feedback loops appear. This represents a failure of compositionality.

This restriction does not appear to have a major practical significance, as it only rules out a small class of useful circuitry [Rie04]. However, from a theoretical point of view, it presents an interesting challenge since compositionality is widely accepted as good theoretical methodology [FS18]. On general principle, we have reason to expect that a compositional theory of digital circuits may lead to more streamlined methods of analysis and verification, which, in time, may lead to improved logical designs. Semantic domains in which circuits with combinational feedback can be interpreted are known [MSB12], but the interpretation given to circuits is not compositional. On the other hand, fully compositional syntactic and categorical accounts of circuits have led to novel operational, rewriting-based, semantics [GJ16; GJL17a].

In the current paper we bring together these two semantic models, denotational and categorical, giving a fully compositional interpretation of digital circuits, including those which may have combinational feedback loops. To be more precise, by 'digital circuits' we primarily understand circuits formed of logical gates and basic memory elements such as latches or D flip-flops of known and fixed delays. But the same machinery can be used to interpret any deterministic circuit that has a clear notion of input and output and which works with discrete signals, such as CMOS transistors operating in saturation mode. What we do not attempt to handle are circuits operating on continuous signals (such as amplifiers) or in continuous time (such as asynchronous circuits).

The semantic domain of interpretation for digital circuits is essentially that of stream functions. A stream is a sequence of digital values, sometimes called 'waveforms' in digital design lingo: they are the input and output of digital circuits. Functions that are representations of circuits have certain characteristics. They are *monotone* and *causal*, properties related to computability by physical devices. Moreover, they specify finitely many possible behaviours, a characteristic of the finite-state nature of digital circuits.

The main technical result of the paper is to show that the stream function interpretation (causal and with finite behaviour) is fully abstract for digital circuits. This means that equality in the equational theory of digital circuits, which is a symmetric traced Cartesian category (also known as a 'dataflow category') augmented with special domain-specific axioms, is equivalent to equality of stream functions (with the stated characteristics). This gives, for the first time, a definitive compositional theory of digital circuits. The new and improved model also allows us to revisit and improve the older operational models, showing that the unpleasant situation of 'unproductive' circuits, i.e. circuits that cannot be evaluated operationally via rewriting, can now be fully avoided. Consequently, any circuit can be syntactically reduced to a (potentially infinite but always periodic) sequence of values.

The mathematical language we use is that of *string diagrams* [JS91], a two dimensional syntax for monoidal categories. Expressing the theory of digital circuits using string diagrams is not only (arguably) rather intuitive, but also technically advantageous [GJL17a]. Particularly helpful is the formal connection between string diagrams and graph rewriting, detailed elsewhere [Bon+22a; Bon+21; Bon+22b; Kay21].

Two technical developments are instrumental in achieving the full abstraction result. First, we lift the well-known formalism of Mealy machines [Mea55] to work with lattices, rather than unstructured data: this allows us to show that digital circuits correspond precisely to stream functions which are monotone, causal, and have finite behaviours. Second, we formulate a version of the Kleene fixpoint theorem to show how circuits with combinational feedback loops can be transformed into circuits without such loops through a finite, globally fixed (and usually small) number of unfoldings of the loop. In retrospect this may appear obvious, as is sometimes the case with semantic insights. However, the normal form which circuits must take in order to make the unfolding possible is only 'obvious' in the string-diagrammatic formulation. It is not at all clear how circuits specified globally and non-compositionally [MSB12] could be unfolded in this way. This is perhaps why this seemingly obvious solution was elusive until now. Finally, we present a syntactic method of relating circuits with the same behaviour by reducing the problem to that of checking a finite number of closed circuits: with this, we achieve full abstraction.

2 Digital circuits

Let us first revisit the categorical semantics of digital circuits [GJ16].

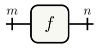
Definition 1 (Circuit signature, value, gate symbol)**.** *A* circuit signature Σ *is a tuple* $(\mathcal{V}, \bullet, \circ, \mathcal{G})$ *where* $\mathcal{V}$ *is a finite set of* values *with distinguished elements* $\bullet, \circ \in \mathcal{V}$*, and* $\mathcal{G}$ *is a finite set of tuples* (g, m) *where* g *is a* gate symbol *and* $m \in \mathbb{N}$ *is its* arity*.*

The distinct elements $\bullet$ and $\circ$ represent a *disconnected wire* (a *lack* of information) and a *short circuit* (*inconsistent* information) respectively. A particularly important signature is that of gate-level circuits, the most common level of abstraction for digital circuits.

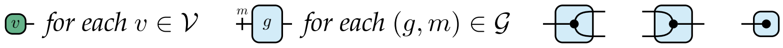
Definition 2 (Gates)**.** *Let* $\mathcal{V}_\star = \{\mathsf{n}, \mathsf{f}, \mathsf{t}, \mathsf{b}\}$*, respectively representing* no *signal, a* false *signal, a* true *signal and* both *signals at once. Let* $\mathcal{G}_\star = \{(\mathrm{AND}, 2), (\mathrm{OR}, 2), (\mathrm{NOT}, 1)\}$*. The signature for* gate-level circuits *is* $\Sigma_\star = (\mathcal{V}_\star, \mathsf{n}, \mathsf{b}, \mathcal{G}_\star)$*.*

2.1 Syntax

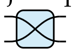
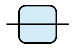
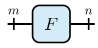
A circuit signature freely generates a monoidal category of *combinational circuits*. We prefer the two dimensional syntax of *string diagrams* [JS91; JSV96; Sel11] to the linear term syntax as it is both intuitive and technically convenient. Diagrams are written left-to-right, with generators represented as boxes, composition as horizontal juxtaposition, and tensor product as vertical juxtaposition. One of the desirable properties of this notation is that structural rules (identity, associativity, functoriality) are absorbed into the diagrams interpreted as graphs up to isomorphism.

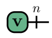
String diagrams are an especially natural language for *props* [Lac04], symmetric monoidal categories with natural numbers as objects and addition as tensor product. A generator $m \to n$ is then drawn as a box with m input wires and n output wires. For compactness, multiple wires may be drawn as one wire: these wires will be notched and labelled:  .

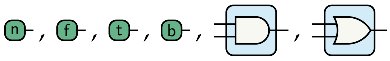
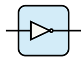
Definition 3 (Combinational circuits)**.** *Given a circuit signature $\Sigma = (\mathcal{V}, \bullet, \circ, \mathcal{G})$, let $\mathbf{CCirc}_\Sigma$ be the symmetric strict monoidal prop generated freely over*



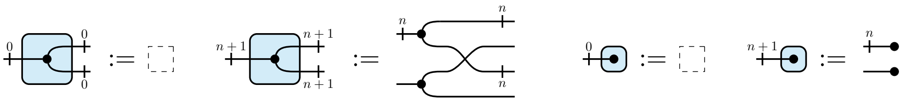
Generators in small dark boxes are *values*, representing the signals that flow through wires. The lighter rectangular generators are *gates*. The last three generators are *structures* for forking, joining and stubbing wires. The reasoning behind the box colours will become clear when we consider the semantics of circuits.

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 . Arbitrary circuit morphisms containing only gates and structures are drawn as light boxes  .

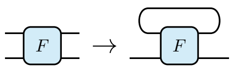
Multiple values in parallel will often be combined into a single box: , where $\mathbf{v}$ is a list of values $[v_0, v_1, \cdots, v_{n-1}]$. When considering an arbitrary signature, the designated $\bullet$ value is drawn as , and as with standard values, multiple $\bullet$ values may be condensed into one.

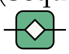
Example 4. The signature $\Sigma_\star$ defines the generators , and .

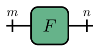
Using the structural generators, forks and discards of arbitrary width are defined as follows:



Joins and values of multiple bits are defined dually. To represent *sequential* circuits we add *delay* and *feedback*.



Definition 5 (Sequential circuits)**.** *Let $\mathbf{SCirc}_\Sigma$ be the category obtained by freely extending $\mathbf{CCirc}_\Sigma$ with a new generator*  *and a trace operator.*

The new generator can be thought of as 'delaying inputs by one tick of the clock'. The addition of a trace makes $\mathbf{SCirc}_\Sigma$ a *symmetric traced monoidal category* [JSV96], or STMC. The axioms of STMCs are standard: they are listed in Appendix A. Sequential circuit morphisms are drawn as square dark boxes:  .

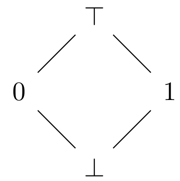
2.2 Semantics

To add computational content to circuits, the signature must be interpreted in a semantic domain.

Definition 6 (Interpretation)**.** *An interpretation of $\Sigma = (\mathcal{V}, \bullet, \circ, \mathcal{G})$ is a tuple $\mathcal{I} = (\mathbf{V}, \mathcal{I}_\mathcal{V}, \mathcal{I}_\mathcal{G})$ where $(\mathbf{V}, \sqsubseteq, \bot, \top)$ is a finite lattice, $\mathcal{I}_\mathcal{V}$ is a function $\mathcal{V} \setminus \{\bullet, \circ\} \to \mathbf{V} \setminus \{\bot, \top\}$, and $\mathcal{I}_\mathcal{G}$ is a map from each $(g, m) \in \mathcal{G}$ to a monotone function $\overline{g} \colon \mathbf{V}^m \to \mathbf{V}$ such that $\overline{g}(\bot^m) = \bot$ and $\overline{g}(\mathbf{v})$ is in the image of $\mathcal{I}_\mathcal{V}$ for all $\mathbf{v} \in \mathbf{V}^m$.*

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

Example 7. Recall the signature $\Sigma_\star = (\mathcal{V}_\star, \mathsf{n}, \mathsf{b}, \mathcal{G}_\star)$ from Definition 2. The values are interpreted in the four value lattice $\mathbf{V}_\star = \{\bot, 0, 1, \top\}$, where $0 \sqcup 1 = \top$ and $0 \sqcap 1 = \bot$. The gates are interpreted using Belnap logic [Bel77]: the truth tables are listed in Figure 1. Let $\mathcal{I}_\star = (\mathbf{V}_\star, \{\mathsf{f} \mapsto 0, \mathsf{t} \mapsto 1\}, \{\mathrm{AND} \mapsto \wedge, \mathrm{OR} \mapsto \vee, \mathrm{NOT} \mapsto \neg\})$. The astute reader may observe that $\wedge$ and $\vee$ are in fact the join and meet of another lattice structure on $\{\bot, 0, 1, \top\}$, in which 1 is the supremum and 0 the infimum.



$\wedge$	$\bot$	0	1	$\top$
$\bot$	$\bot$	0	$\bot$	0
0	0	0	0	0
1	$\bot$	0	1	$\top$
$\top$	0	0	$\top$	$\top$

$\vee$	$\bot$	0	1	$\top$
$\bot$	$\bot$	$\bot$	1	1
0	$\bot$	0	1	$\top$
1	1	1	1	1
$\top$	1	$\top$	1	$\top$

$\neg$	
$\bot$	$\bot$
1	0
0	1
$\top$	$\top$

Figure 1: The lattice structure on $\mathbf{V}_\star$, and the truth tables of Belnap logic [Bel77].

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

Semantics for digital circuits are described in terms of *streams*, infinite sequences of values over time. Given a set X, the set of streams of X is denoted X^ω. For a stream $\sigma \in X^\omega$, its kth element is written $\sigma(k)$. A stream X^ω is equipped with two operations: the *initial value* $\mathsf{i}(-)\colon X^\omega \to X$, which produces the 'head' of the stream; and *stream derivative* $\mathsf{d}(-)\colon X^\omega \to X^\omega$, a function producing the 'tail' of the stream, defined as $\mathsf{d}(\sigma)(i) = \sigma(i+1)$.

In particular, circuits can be interpreted as *stream functions*, which consume streams as input and produce streams as output. As with regular streams, given a stream function f and some input stream σ, the kth element of its output stream is written $f(\sigma)(k)$.

Example 8. Given an interpretation $\mathcal{I}$, some useful stream functions are as follows. For a value $v \in \mathbf{V}$, the function $\mathsf{val}_v\colon (\mathbf{V}^0)^\omega \to \mathbf{V}^\omega$ is defined as $\mathsf{val}_v(\bullet)(0) := v$, $\mathsf{val}_v(\bullet)(i+1) := \bot$. For a gate $g\colon \mathbf{V}^m \to \mathbf{V} \in \mathcal{I}_\mathcal{G}$, the function $\mathsf{gate}_g\colon (\mathbf{V}^m)^\omega \to \mathbf{V}^\omega$ is defined as $\mathsf{gate}_g(\sigma)(i) := g(\sigma(i))$. Finally there are functions $\mathsf{fork}\colon \mathbf{V}^\omega \to (\mathbf{V}^2)^\omega$ where $\mathsf{fork}(\sigma)(k) := (\sigma(k), \sigma(k))$, $\mathsf{join}\colon (\mathbf{V}^2)^\omega \to \mathbf{V}^\omega$ where $\mathsf{join}(\sigma,\tau)(k) := (\sigma(k) \sqcup \tau(k))$, $\mathsf{stub}\colon \mathbf{V}^\omega \to (\mathbf{V}^0)^\omega$ where $\mathsf{stub}(\sigma)(k) := (\bullet)$ and $\mathsf{shift}\colon \mathbf{V}^\omega \to \mathbf{V}^\omega$ where $\mathsf{shift}(\sigma)(0) := \bot$, $\mathsf{shift}(\sigma)(i+1) := \sigma(i)$.

Definition 9. *For an interpretation $\mathcal{I} = (\mathbf{V}, \mathcal{I}_\mathcal{V}, \mathcal{I}_\mathcal{G})$, let $\mathbf{Stream}_\mathcal{I}$ be the prop freely generated over val_v for each $v \in \mathcal{V}$, gate_g for each $(g,m) \in \mathcal{G}$, and the stream functions fork, join, stub and shift. Composition and tensor are by function composition and Cartesian product; the symmetry swaps input streams.*

For $\mathbf{Stream}_\mathcal{I}$ to be a suitable semantic domain for sequential circuits, it must be traced. Finding such an construction is not necessarily possible for an arbitrary stream function, but $\mathbf{Stream}_\mathcal{I}$ enjoys certain properties that mean a suitable traced morphism exists. These properties will now be examined.

Definition 10 (Causal stream function [BRS08])**.** *A stream function $f\colon M^\omega \to N^\omega$ is* causal *if for all $i \in \mathbb{N}$ and $\sigma, \tau \in M^\omega$, it holds that $f(\sigma)(i) = f(\tau)(i)$ whenever $\sigma(j) = \tau(j)$ for all $0 \leq j \leq i$.*

Effectively, if a stream function is causal, then the ith element of the output stream depends only on the first $i+1$ inputs. When stream functions are causal, the notion of initial value and stream derivative from 'ordinary' streams are extendable to stream functions.

Definition 11 (Functional stream derivative [Rut06])**.** *Suppose $f\colon M^\omega \to N^\omega$ is a causal stream function and let $a \in M$. The* initial output of f on input a *is $f[a] = \mathsf{i}(f(a :: \sigma)) \in N$ for any $\sigma \in M^\omega$. The* functional stream derivative of f on input a *is a function $f_a\colon M^\omega \to N^\omega$ given by $f_a(\sigma) = \mathsf{d}(f(a :: \sigma))$.*

The causality of f ensures $f[a]$ does not depend on the choice of σ. f_a acts as f would 'had it seen a first'. Using causality, the notion of *monotonicity* of stream functions can also be expressed.

Definition 12 (Monotone stream function)**.** *Let M and N be partially ordered sets. Then, a causal stream function $f\colon M^\omega \to N^\omega$ is monotone if, for any $a \in M$, $a \mapsto f[a]$ is monotone, and f_a is a monotone stream function.*

Lemma 13. *Causality and monotonicity are preserved by composition and product.*

Proof. For causality, if the ith element of two stream functions f and g only depends on the first $i+1$ elements of the input, then so will their composition. For monotonicity, the composition of two monotone functions is also monotone, so the initial output of two composed stream functions with monotone initial outputs will also be monotone. $\square$

Lemma 14. *All stream functions in $\mathbf{Stream}_\mathcal{I}$ are causal and monotone.*

Proof. Since causality and monotonicity is preserved by function composition and disjoint union, it suffices to check that the generators are causal and monotone. The value streams have no inputs, so are vacuously causal and monotone. The ith element of the gate and structure streams depend solely on the ith element of the input, so they are causal. Moreover, their behaviour at each tick is a monotone function by definition. The 0th element of the shift stream does not depend on σ and the $i+1$th element depends on the ith element of the input, so δ is causal. Given an element $v \in \mathbf{V}$, $\mathsf{shift}[v] = \bot$, which is monotone, and shift_v is a stream function that outputs v as its first element regardless of the first input: therefore this is also a monotone stream function. □

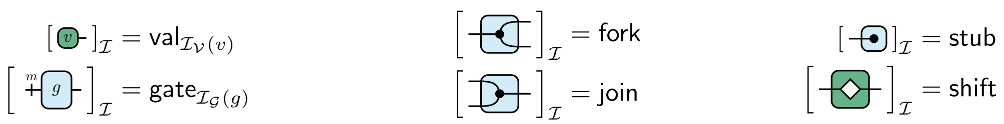
These properties are enough to define a trace on $\mathbf{Stream}_{\mathcal{I}}$. Given tuples $u \in X^m, v \in X^n$, we write $u +\!\!+ v \in X^{m+n}$ for their *concatenation*: the tuple containing the elements of u followed by the elements of v. Abusing notation, given two streams $\sigma \in (X^m)^\omega, \tau \in (X^n)^\omega$, we also write $\sigma +\!\!+ \tau \in (X^{m+n})^\omega$ for their pointwise concatenation. For a stream function $\sigma \in (X^{m+n})^\omega$, we write $\pi_m(\sigma) \in (X^m)^\omega$ for the stream of tuples containing the first m elements of tuples in σ, and $\pi_n(\sigma)$ for the stream of tuples containing the last n elements.

Proposition 15. *Let $f\colon (\mathbf{V}^{x+m})^\omega \to (\mathbf{V}^{x+n})^\omega$ be a morphism in $\mathbf{Stream}_{\mathcal{I}}$. For each $i \in \mathbb{N}$ and $\sigma \in (\mathbf{V}^m)^\omega$, there is a monotone endofunction on $(\mathbf{V}^x)^\omega$ given by $\tau \mapsto \pi_x(f(\tau +\!\!+ \sigma))$; let μ_σ be the least fixed point of this function. Then a trace $\mathrm{Tr}^x(f)\colon (\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$ is defined by letting $(\mathrm{Tr}^x(f))(\sigma)$ be $\pi_n(f(\mu_\sigma +\!\!+ \sigma))$.*

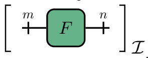
Proof. Since the stream functions are monotone, the initial value of each stream derivative is a monotone function, so they will have a least fixed point. The axioms of traced categories can be shown to hold with this construction. □

Stream functions in $\mathbf{Stream}_{\mathcal{I}}$ are the semantics for morphisms in $\mathbf{SCirc}_\Sigma$. A (*traced*) *prop morphism* is a strict (traced) symmetric monoidal functor between props. The props involved are freely generated, so it suffices to define prop morphisms solely on the generators.

Definition 16. *Let $[-]_{\mathcal{I}} : \mathbf{SCirc}_\Sigma \to \mathbf{Stream}_{\mathcal{I}}$ be the traced prop morphism with its action defined as:*



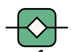
3 Mealy machines

There is one facet of sequential circuits that we have not yet examined. Since digital circuits are constructed of a finite number of components, they define a finite amount of behaviour. That is to say, given an input stream σ, the stream function  must have *finitely many stream derivatives*. To show that this is the case in $\mathbf{Stream}_{\mathcal{I}}$, we view circuits through the lens of *Mealy machines* [Mea55]: causal stream functions form their *final coalgebra* [Rut06].

Definition 17 (Mealy machine [Mea55])**.** *For finite sets M and N, a (finite)* Mealy machine *with interface (M, N) is a tuple (S, f) where S is a finite* state space *and $f\colon S \to (N \times S)^M$ is the* Mealy function*.*

The sets M and N are the *input* and *output* spaces of the machine. Given a state $s \in S$ and input $m \in M$, the Mealy function f produces a pair $f(s)(a) = \langle n, s' \rangle$. Following [BRS08], we write $f(s)(a) = \langle s[a], s_a \rangle$ and call $s[a]$ the *output on input a* and s_a the *transition on input a*.

3.1 Mealy machines for sequential circuits

There are two components in circuits that determine state: values  and delays . Each of these components 'contains' a value in $\mathbf{V}$ at any point in time: therefore the state space of a corresponding Mealy machine will be a power of $\mathbf{V}$. With this in mind, the traditional Mealy machine definition can be specialised for the context of sequential circuits.

Definition 18. *Let $\mathcal{I} = (\mathbf{V}, \mathcal{I}_{\mathcal{V}}, \mathcal{I}_{\mathcal{G}})$ be an interpretation. The (non-traced) prop $\mathbf{Func}_{\mathcal{I}}$ is freely generated over*

- *the functions $\overline{g}\colon \mathbf{V}^m \to \mathbf{V}$ for each $(g, m) \in \mathcal{I}_{\mathcal{G}}$; and*
- *the functions $(x) \mapsto (x, x)$, $(x, y) \mapsto x \sqcup y$ and $(x) \mapsto (\bullet)$.*

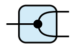
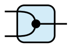
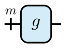
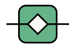
Generator	$s[x]$	s_x	$\bar{s}$	Generator	$s[x]$	s_x	$\bar{s}$
	$s(0)$	$(s(1), s(1))$	$(v, \bot)$		$(x(0), x(0))$	$(\bullet)$	$(\bullet)$
	$(x(0) \sqcup x(1))$	$(\bullet)$	$(\bullet)$		$(\bullet)$	$(\bullet)$	$(\bullet)$
	$(\bar{g}(x))$	$(\bullet)$	$(\bullet)$		s	x	$(\bot)$

Figure 2: Action of the prop morphism $[\![-]\!]_{\mathcal{I}}$ on generators of $\mathbf{SCirc}_\Sigma$.

Definition 19 ($\mathcal{I}$-Mealy machines)**.** *For an interpretation $\mathcal{I}$ and $m, n \in \mathbb{N}$, an* $\mathcal{I}$-Mealy machine *over interface (m, n) is a tuple $(\mathbf{V}^r, f, \bar{s})$, where $r \in \mathbb{N}$, $f \colon \mathbf{V}^r \to (\mathbf{V}^n \times \mathbf{V}^r)^{\mathbf{V}^m}$ is a morphism in $\mathbf{Func}_{\mathcal{I}}$, and $\bar{s} \in \mathbf{V}^r$ is a specified* initial state*.*

Definition 20. *Let $\mathbf{Mealy}_{\mathcal{I}}$ be the traced Cartesian prop with morphisms $m \to n$ the $\mathcal{I}$-Mealy machines over interface (m, n). The operations are defined below.*

- *Composition is the* cascade product*, defined for machines $A \colon m \to n = (\mathbf{V}^r, \langle s[a]^A, (s_a)^A\rangle, \bar{s}^A)$ and $B \colon n \to p = (\mathbf{V}^{r'}, \langle s[a]^B, (s_a)^B\rangle, \bar{s}^A)$ as*

$$(\mathbf{V}^{r+r'}, (s^A \mathbin{+\!\!+} s^B, v) \mapsto \langle s'[s[v]^A]^B, (s_v)^A \mathbin{+\!\!+} (s_{s[v]^A})^B\rangle, \bar{s}^A \mathbin{+\!\!+} \bar{s}^B).$$

- *Tensor is the* direct product*, defined for machines $A \colon m \to n = (\mathbf{V}^r, \langle s[a]^A, (s_a)^A\rangle, \bar{s}^A)$ and $B \colon p \to q = (\mathbf{V}^{r'}, \langle s[a]^B, (s_a)^B\rangle, \bar{s}^B)$ as*

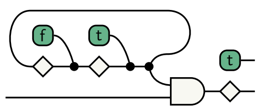
$$(\mathbf{V}^{r+r'}, (s^A \mathbin{+\!\!+} s^B, v) \mapsto (s[v]^A \mathbin{+\!\!+} s'[v']^B, (s_v)^A \mathbin{+\!\!+} (s'_{v'})^B), \bar{s}^A \mathbin{+\!\!+} \bar{s}^B).$$

- *The identity machine on m is defined as $(\mathbf{V}^0, (\bullet, v) \mapsto (\bullet, v), \bullet)$.*
- *The symmetry machine on m and n is defined as $(\mathbf{V}^0, (\bullet, v \mathbin{+\!\!+} w) \mapsto (w \mathbin{+\!\!+} v))$.*
- *The copy machine is defined as $(\mathbf{V}^0, (\bullet, v) \mapsto (v \mathbin{+\!\!+} v, \bullet), \bullet)$.*
- *The discard machine is defined as $(\mathbf{V}^0, (\bullet, v) \mapsto (\bullet, \bullet), \bullet)$.*
- *Given a machine $(A \colon x + m \to x + n = (S^A, \langle s[a]^A, (s_a)^A\rangle, \bar{s}^A))$, let $\mathsf{fix}_{s,v}$ be the least fixpoint of the function $a \mapsto \pi_x(s[a \mathbin{+\!\!+} v]^A)$ for given input v. Then the trace of A is defined as*

$$(S^A, (s, v) \mapsto (\pi_n(s[\mathsf{fix}_{s,v} \mathbin{+\!\!+} v]^A), (s_{(\mathsf{fix}_{s,v} +\!\!+ v)})^A))$$

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

Definition 21. *Let $[\![-]\!]_{\mathcal{I}} : \mathbf{SCirc}_\Sigma \to \mathbf{Mealy}_{\mathcal{I}}$ be the traced prop morphism with action shown in Fig. 2.*

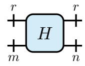
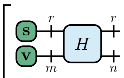
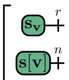
Example 22. Consider the circuit . This contains three values and three delays: each value contributes two elements to the state and each delay contributes one, so the state space is $\mathbf{V}^9$. One possible initial state is $(\mathsf{f}, \bot, \bot, \mathsf{t}, \bot, \bot, \mathsf{t}, \bot, \bot)$: its corresponding Mealy function is $f(r_0, \cdots, r_8)(x) = \langle (r_6, r_8), (r_1, r_1, r_2 \sqcup r_3, r_4, r_4, r_0 \sqcup r_1, r_7, r_7, (r_2 \sqcup r_3) \wedge x)\rangle$.

Mealy machines have interpretations as stream functions. This is expressed coalgebraically: any Mealy machine (S, f) with interface (M, N) is a coalgebra of the functor $Y \colon \mathbf{Set} \to \mathbf{Set}$, defined as $YX = (N \times S)^M$. A homomorphism h between two Mealy machines (S, f) and (T, g) with interface (M, N) is a function $f \colon S \to T$ preserving outputs and transitions, i.e. $hs[a] = s[a]$ and $h(s_a) = h(s)_a$. Stream functions form the *final coalgebra* for Mealy machines.

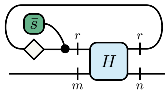
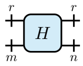
Proposition 23 (Proposition 2.2, [Rut06])**.** *Fix an input space M and output space N. Let Γ be the set of causal stream functions $f \colon M^\omega \to N^\omega$, and let $\nu \colon \Gamma \to (N \times \Gamma)^M$ be the function defined as $\nu(f)(m) = \langle f[m], f_m\rangle$. Then for every Mealy machine (S, f) with interface (M, N), there exists a unique homomorphism $\hat{h} \colon (S, f) \to (\Gamma, \nu)$.*

Proof. Given an initial state $\bar{s}$, the unique stream function is defined for an input stream σ as $\bar{s}[\sigma(0)] :: \bar{s}_{\sigma(0)}[\sigma(1)] :: (\bar{s}_{\sigma(0)})_{\sigma(1)}[\sigma(2)] :: \cdots$. □

Since an $\mathcal{I}$-Mealy machine $A = (\mathbf{V}^r, f, \bar{s})$ is equipped with a start state $\bar{s}$, we write $!A := \hat{h}(\bar{s})$ for the map to the stream function generated from $\bar{s}$ for Mealy machine $(\mathbf{V}^r, f)$. We will now show that $!(-)$ is a prop morphism $\mathbf{Mealy}_{\mathcal{I}} \to \mathbf{Stream}_{\mathcal{I}}$ by defining a translation back from Mealy machines to circuits.

Lemma 24. *Given $\mathcal{I}$-Mealy machine $(\mathbf{V}^r, \bar{s}, (s, v) \mapsto \langle s[v], s_v\rangle)$, there exists a combinational circuit*  $\in$ $\mathbf{SCirc}_{\Sigma}$ *such that* $\left[\right.$  $\left.\right]_{\mathcal{I}} = \left[\right.$  $\left.\right]_{\mathcal{I}}$.

Proof. The Mealy function is a morphism in $\mathbf{Func}_{\mathcal{I}}$, so is decomposable into generators in $\mathcal{I}$. □

Definition 25. *Let $\langle\!\langle - \rangle\!\rangle_{\mathcal{I}} \colon \mathbf{Mealy}_{\mathcal{I}} \to \mathbf{SCirc}_{\Sigma}$ be the traced prop morphism defined for an $\mathcal{I}$-Mealy machine* $(\mathbf{V}^r, f, \bar{s})$ *as*  *, where*  *is defined as in Lemma 24.*

Theorem 26. $!(-) = [-]_{\mathcal{I}} \circ \langle\!\langle - \rangle\!\rangle_{\mathcal{I}}$.

Proof. The interpretation of $\langle\!\langle - \rangle\!\rangle_{\mathcal{I}}$ as a stream function is the fixpoint of the function

$$g(\tau +\!\!+ \sigma)(i) = \begin{cases} \langle T, O\rangle(\bar{s}, \sigma(0)) & i = 0 \\ \langle T, O\rangle(\tau(k), \sigma(k+1)) & i = k+1 \end{cases}$$

which is precisely the stream function $O :: T \circ O :: T^2 \circ O :: \cdots$ obtained by using $!(-)$. □

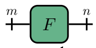
Corollary 27. *$!(-)$ is a prop morphism $\mathbf{Mealy}_{\mathcal{I}} \to \mathbf{Stream}_{\mathcal{I}}$.*

It is now possible to show the final characteristic of stream functions in $\mathbf{Stream}_{\mathcal{I}}$.

Lemma 28. *Any stream in the image of $!(-)$ has finitely many stream derivatives.*

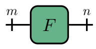
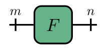
Proof. Given an $\mathcal{I}$-Mealy machine A, the state set of A is finite by definition. $!A$ has at most as many stream derivatives as A does states, since $!(-)$ preserves transitions. □

Theorem 29. $[-]_{\mathcal{I}} = !(-) \circ [\![-]\!]_{\mathcal{I}}$.

Proof. By induction on the structure of  . The generators, composition and tensor are trivial. The trace on Mealy machines and streams is constructed by computing the least fixed point, so they are equal. □

Corollary 30. *Any stream in the image of $[-]_{\mathcal{I}}$ has finitely many stream derivatives.*

4 Circuit synthesis

We have established that each sequential circuit specified syntactically has a corresponding monotone causal stream function with finite stream derivatives. Now we consider the reverse direction: given a function $f : (\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$, is there a recipe for a circuit  such that $\left[\right.$  $\left.\right]_{\mathcal{I}} = f$?

4.1 Synthesising a Mealy machine

As mentioned above, a causal stream function $f \colon M^\omega \to N^\omega$ is in fact a Mealy machine with interface (M, N). Given such a function f, a minimal Mealy machine is obtainable.

Corollary 31 (Corollary 2.3, [Rut06])**.** *If $f \colon M^\omega \to N^\omega$ is a causal stream function, let S be the least set of causal stream functions including f and closed under stream derivatives: i.e. for all $h \in S$ and $a \in M$, $h_a \in S$. Then the Mealy machine $\mathbf{S_f} = (S, g)$ where $g(h)(a) = \langle h[a], h_a\rangle$, has the smallest state space of Mealy machines with the property $!\mathbf{S_f} = f$.*

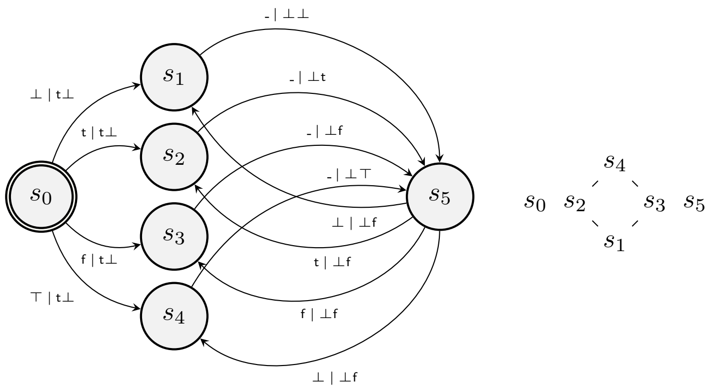


Figure 3: On the left, the Mealy machine from Example 32, where a transition label $v \,|\, w$ indicates an input v and an output w. On the right, the corresponding state ordering.

Proof. Since S is generated from the function f and is the *smallest* possible set, there are no unreachable states in S and no two states can 'share the same behaviour'. $\square$

Example 32. Recall the circuit from Example 22. The corresponding stream function $f\colon \mathbf{V}^\omega \to \mathbf{V}^\omega$ is defined for input σ as

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

We shall now recover a (finite) Mealy machine from this function. From the first state $s_0 = f$, the output is $f[v] = (\mathsf{t}, v)$. The state $s_1 = f_\bot$ is defined as $(s_1)(\sigma) = (f(\bot :: \sigma)) = (\bot, \mathsf{f}), (\bot, \sigma(0)), \cdots$. $s_2 = f_\mathsf{t}$, $s_3 = f_\mathsf{f}$ and $s_4 = f_\top$ are defined similarly but are distinct states. The output of states $1 - 4$ is always $(\bot, \mathsf{f})$ 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 $(s_5)(\sigma) = d^3(f(\bot :: v :: \bot :: \sigma)) = (\bot, \mathsf{f}) :: (\bot, \sigma(0)) :: \cdots$: this is the same as s_1. Similarly, inputting t, f and $\top$ produces the states s_2, s_3 and s_4 respectively. So we have fully specified the Mealy machine: it is illustrated in Figure 3.

4.2 Synthesising a monotone Mealy machine

To obtain an $\mathcal{I}$-Mealy machine, and subsequently a sequential circuit, each state in $\mathbf{S_f}$ must be *encoded* as a power of $\mathbf{V}$. This is a standard procedure in circuit design [KJ09], but in our context the encoding must also respect monotonicity.

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.

Definition 33. *Let M be a set and let N be a partially ordered set; let $f, g\colon M^\omega \to N^\omega$ be two causal stream functions. We say $f \preceq g$ if $f(\sigma)(i) \leq_N g(\sigma)(i)$ for all $\sigma \in M^\omega$ and $i \in \mathbb{N}$.*

Lemma 34. *The relation $\preceq$ is a partial order.*

For a Mealy machine $\mathbf{S_f}$ derived as in Corollary 31, its states are causal stream functions $(\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$, so they inherit this ordering. In an $\mathcal{I}$-Mealy machine, the Mealy function must be monotone with respect to this ordering.

Example 35. The state ordering for the Mealy machine derived in Example 32 is shown in Figure 3.

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.

Proposition 36. *Let $f, g\colon M^\omega \to N^\omega$ be monotone stream functions such that $f \preceq g$. Then $f[a] \leq_N g[a]$ and $f_a \preceq g_a$ for all $a \in M$.*

Proof. Let $\sigma \in M^\omega$ be arbitrary. By Definition 11 and the fact that $\mathsf{i}(-)$ is monotone, $f[a] = \mathsf{i}(f(a :: \sigma)) \leq_N \mathsf{i}(g(a :: \sigma)) = g[a]$. Similarly, $f_a(\sigma) = \mathsf{d}(f(a :: \sigma)) \leq_{N^\omega} \mathsf{d}(g(a :: \sigma)) = g_a(\sigma)$ by definitions and the monotonicity of stream derivative, so $f_a \preceq g_a$. □

Proposition 37. *Let $f\colon M^\omega \to N^\omega$ be a monotone causal function for partially ordered sets M and N: given the corresponding Mealy machine $\mathbf{S_f} = (S, g)$ defined as in Proposition 31, the Mealy function g is monotone.*

Proof. Proposition 36 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 $a \leq_M a'$. Since h is monotone, $h[a] = \mathsf{i}(h(a :: \sigma)) \leq_N \mathsf{i}(h(a' :: \sigma)) = h[a']$, and similarly for the transition function. As these functions are monotone in both components, they are monotone overall. □

Definition 38 (State assignment)**.** *Let $S = \{s_0, s_1, \cdots, s_{r-1}\}$ be a state space with an ordering $\preceq$. Then, the state assignment $\gamma\colon S \to \mathbf{V}^r$ is defined for $\gamma(s_j)(i)$ as $\top$ if $s_j \preceq s_i$ and $\perp$ otherwise.*

Lemma 39. *In the context of Definition 38, if $s \preceq s'$, then $\gamma(s) \sqsubseteq \gamma(s')$.*

4.3 Synthesising an $\mathcal{I}$-Mealy machine

There is one final point to consider: to reconstruct an $\mathcal{I}$-Mealy machine from a Mealy machine $\mathbf{S_f}$, the transition and output functions must be expressible as morphisms in $\mathbf{Func}_\mathcal{I}$.

Definition 40 (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 41. The Belnap functions $\{\wedge, \vee, \neg\}$ from Example 7 are functionally complete.

Proposition 42. *Given functionally complete $\mathcal{I}$ and a Mealy machine $\mathbf{S_f} = (S, g)$ derived as above, there exists $\mathbf{v} \in \mathbf{V}^x$ and $h\colon x + r + m \to x + r + n \in \mathbf{Func}_\mathcal{I}$ such that for any $\mathbf{s} \in \mathbf{V}^r$ and $\mathbf{a} \in \mathbf{V}^m$, if $g(\mathbf{s})(\mathbf{a}) = (\mathbf{s}', \mathbf{b})$ then $h(\mathbf{v} \mathbin{+\!\!+} \gamma(s) \mathbin{+\!\!+} \mathbf{a}) = (\mathbf{v} \mathbin{+\!\!+} \gamma(s'), \mathbf{b})$.*

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

Remark 43. Since the Mealy function must be combinational, the additional values $\mathbf{v}$ are used to define constants.

The translation between streams and Mealy machines will now be formalised.

Definition 44. *Given functionally complete $\mathcal{I}$ and an $\mathcal{I}$-circuit function f, let $\mathbf{S_f}$ be its minimal Mealy machine and let $\mathbf{v}$ and h be defined as in Proposition 42. Let $\gamma^\star(S)$ be the pointwise application of $\mathbf{v} \mathbin{+\!\!+} \gamma(-)$ to S. Then, the prop morphism $\langle - \rangle_\mathcal{I} : \mathbf{Stream}_\mathcal{I} \to \mathbf{Mealy}_\mathcal{I}$ is defined as $\langle f \rangle_\mathcal{I} = (\gamma^\star(S), h, \gamma(f))$.*

There are multiple orders in which the state space can be encoded. However, the behaviour of the resulting Mealy machine does not depend on this, as the following theorem shows.

Theorem 45. *For functionally complete $\mathcal{I}$, $!(-) \circ \langle - \rangle_\mathcal{I} = \mathsf{id}_{\mathbf{Stream}_\mathcal{I}}$.*

Proof. Given a stream function f and input stream σ, let $h = \langle f[a], f_a \rangle$ be the derived Mealy function. Then the stream function $!\,\langle f \rangle_\mathcal{I}$ is equal to

$$\gamma(\bar{s})[\sigma(0)] :: \gamma(\bar{s})_{\sigma(0)}[\sigma(1)] :: \gamma(\bar{s})_{\sigma(0)_{\sigma(1)}}[\sigma(2)] :: \cdots.$$

The Mealy function h acts as $h(\gamma(f))(a) = (f[a], \gamma(f_a))$ by Proposition 42. Therefore

$$\begin{aligned}
\gamma(\bar{s})[\sigma(0)] &= f[\sigma(0)] = f_0(\sigma(0)) \\
\gamma(\bar{s})_{\sigma(0)}[\sigma(1)] &= f_{\sigma(0)}[\sigma(1)] = f_1(\sigma(0), \sigma(1)) \\
\gamma(\bar{s})_{\sigma(0)_{\sigma(1)}}[\sigma(2)] &= f_{\sigma(0),\sigma(1)}[\sigma(2)] = f_2(\sigma(0), \sigma(1), \sigma(2))
\end{aligned}$$

and so on. So this is the original stream function f. □

This means that any function in $\mathbf{Stream}_\mathcal{I}$ can be modelled by an $\mathcal{I}$-Mealy machine. All the components are now in place to state the final result of this section.

Theorem 46. *Define two prop morphisms*

$$\phi\colon \mathbf{SCirc}_\Sigma \to \mathbf{Stream}_\mathcal{I} := \,!(-) \circ [\![-]\!]_\mathcal{I} \text{; and}$$
$$\psi\colon \mathbf{Stream}_\mathcal{I} \to \mathbf{SCirc}_\Sigma := \langle\!\langle - \rangle\!\rangle_\mathcal{I} \circ \langle - \rangle_\mathcal{I}\,.$$

Then the following statements hold:

$$[-]_\mathcal{I} \circ \psi \circ \phi = [-]_\mathcal{I}$$
$$\phi \circ \psi = \mathsf{id}_{\mathbf{Stream}_\mathcal{I}}$$

Proof. First we show the former:

$$\begin{aligned}
[-]_\mathcal{I} \circ \psi \circ \phi &= [-]_\mathcal{I} \circ \langle\!\langle - \rangle\!\rangle_\mathcal{I} \circ \langle - \rangle_\mathcal{I} \circ !(-) \circ [\![-]\!]_\mathcal{I} && \text{(by definition)}\\
&= !(-) \circ \langle - \rangle_\mathcal{I} \circ !(-) \circ [\![-]\!]_\mathcal{I} && \text{(by Theorem 26)}\\
&= !(-) \circ [\![-]\!]_\mathcal{I} && \text{(by Theorem 45)}\\
&= [-]_\mathcal{I} && \text{(by Theorem 26)}
\end{aligned}$$

Now the latter:

$$\begin{aligned}
\phi \circ \psi &= !(-) \circ [\![-]\!]_\mathcal{I} \circ \langle\!\langle - \rangle\!\rangle_\mathcal{I} \circ \langle - \rangle_\mathcal{I} && \text{(by definition)}\\
&= [-]_\mathcal{I} \circ \langle\!\langle - \rangle\!\rangle_\mathcal{I} \circ \langle - \rangle_\mathcal{I} && \text{(by Theorem 29)}\\
&= !(-) \circ \langle - \rangle_\mathcal{I} && \text{(by Theorem 26)}\\
&= \mathsf{id}_{\mathbf{Stream}_\mathcal{I}} && \text{(by Theorem 45)}
\end{aligned}$$

□

This confirms that $\mathbf{Stream}_\mathcal{I}$, a prop with monotone causal stream functions with finitely many stream derivatives, is a suitable semantic domain for sequential circuits.

5 Equational reasoning

When given two circuits, it is common to ask if they have the same *input-output behaviour*, i.e. if their corresponding stream functions are equal.

Definition 47 (Extensional equivalence)**.** *Two sequential circuits* $\overset{m}{-}\boxed{F}\overset{n}{-}$ *and* $\overset{m}{-}\boxed{G}\overset{n}{-}$ *are* extensionally equivalent, *written* $\overset{m}{-}\boxed{F}\overset{n}{-} \approx_\mathcal{I} \overset{m}{-}\boxed{G}\overset{n}{-}$*, if* $\left[\overset{m}{-}\boxed{F}\overset{n}{-}\right]_\mathcal{I} = \left[\overset{m}{-}\boxed{G}\overset{n}{-}\right]_\mathcal{I}$.

As these stream functions have finitely many stream derivatives, to check if two streams are equal we only need to check that the output of the function are equal for a certain number of elements.

Proposition 48 ([GJL17b])**.** *Two sequential circuits* $\overset{m}{-}\boxed{F}\overset{n}{-}$ *and* $\overset{m}{-}\boxed{G}\overset{n}{-}$ *containing no more than* k *delay generators are extensionally equivalent if and only if* $\left[\overset{m}{-}\boxed{F}\overset{n}{-}\right]_\mathcal{I}(\sigma)(i) = \left[\overset{m}{-}\boxed{G}\overset{n}{-}\right]_\mathcal{I}(\sigma)(i)$ *for all* $\sigma \in (\mathbf{V}^m)^\omega$ *and* $i < |\mathbf{V}|^k + 1$.

This establishes a superexponential upper bound for checking if two circuits specified syntactically are extensionally equivalent. An alternative, more efficient, approach is to work *equationally*.

Definition 49. *Let* $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ *be the category obtained by quotienting* $\mathbf{SCirc}_\Sigma$ *by* $\approx_\mathcal{I}$.

Corollary 50. $\mathbf{SCirc}_{\Sigma,\mathcal{I}} \cong \mathbf{Stream}_\mathcal{I}$.

Equations are identities that hold in the category $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$. Given a set of equations $\mathcal{E}$, we write $\overset{m}{-}\boxed{F}\overset{n}{-} =_\mathcal{E} \overset{m}{-}\boxed{G}\overset{n}{-}$ if $\overset{m}{-}\boxed{F}\overset{n}{-}$ is equal to $\overset{m}{-}\boxed{G}\overset{n}{-}$ by equations in $\mathcal{E}$. As we are using string diagrams, the axioms of STMCs are 'absorbed' into the notation and hold by moving wires and boxes around. To show we are applying these axioms, we write $=_{ST}$.

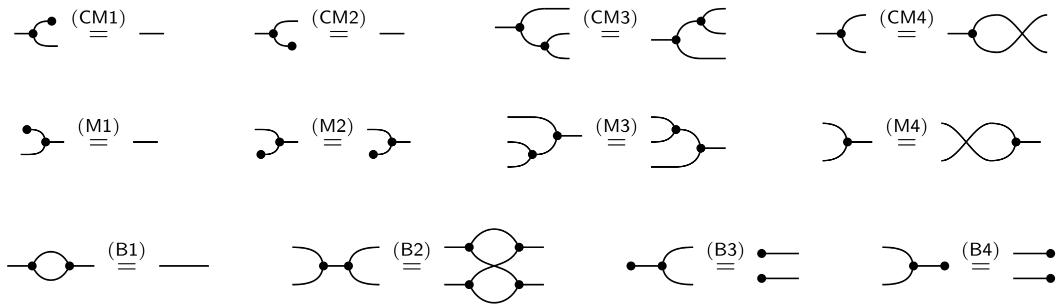
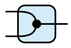
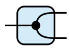


Figure 4: Set $\mathcal{B}$ of *bialgebra* equations.

Figure 5: Set $\mathcal{A}$ of *Cartesian naturality* equations.

5.1 Algebraic structure

One immediate consequence of quotienting by $\approx_{\mathcal{I}}$ is that $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ inherits the structure of $\mathbf{Stream}_{\mathcal{I}}$. The first observation to make is that ( ,  ,  , ) forms a *bialgebra*.

Lemma 51. *The set of equations $\mathcal{B}$ shown in Fig. 4 hold for any interpretation $\mathcal{I}$.*

Proof. The equations are all combinational so it it is a simple exercise to consider the corresponding stream equations at an arbitrary tick. □

Remark 52. Note that one can generalise the equations in $\mathcal{B}$ to forks and joins of arbitrary width easily by repeatedly applying the versions for one wire.

The fork and stub generator inherit some stronger structure: they are *natural*. To show this, we must look at a particular property of $\mathbf{Stream}_{\mathcal{I}}$.

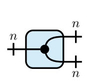
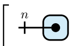
Definition 53 (Cartesian prop)**.** *A prop is* Cartesian *if its tensor product is given by the Cartesian product.*

Any Cartesian category is equipped with families of natural maps called the *diagonal* $\Delta_A \colon A \to A \otimes A$ for copying data, and the *discard* $\diamond_A \colon A \to I$ for deleting data. The naturality of the discard implies that the unit object is *terminal*.

Proposition 54. $\mathbf{Stream}_{\mathcal{I}}$ *is Cartesian with the diagonal map* $\Delta_n \colon (\mathbf{V}^n)^\omega \to (\mathbf{V}^{n+n})^\omega$ *defined as* $\Delta_n(\sigma) = \sigma +\!\!+ \sigma$ *and the discard map* $\diamond_n \colon (\mathbf{V}^n)^\omega \to (\mathbf{V}^0)^\omega$ *defined as* $\diamond_n(\sigma) = (\bullet)^\omega$.

Proof. The tensor in $\mathbf{Stream}_{\mathcal{I}}$ is the Cartesian product, so Δ_n is natural. The discard $\diamond_n$ is natural because 0 is the terminal object in $\mathbf{Stream}_{\mathcal{I}}$. □

Theorem 55. *The set $\mathcal{A}$ of algebraic equations listed in Fig. 5 hold for any interpretation $\mathcal{I}$.*

Proof. Since $\mathbf{Stream}_{\mathcal{I}}$ is Cartesian, all we need to do is show that $\left[\!\!\right.$  $\left.\!\!\right]_{\mathcal{I}} = \Delta_n$ and $\left[\!\!\right.$  $\left.\!\!\right]_{\mathcal{I}} = \diamond_n$. This is by a simple induction over n. □

Corollary 56. $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ *is a Cartesian category.*

A category that is Cartesian and traced is known as a *dataflow* category [CŞ90].

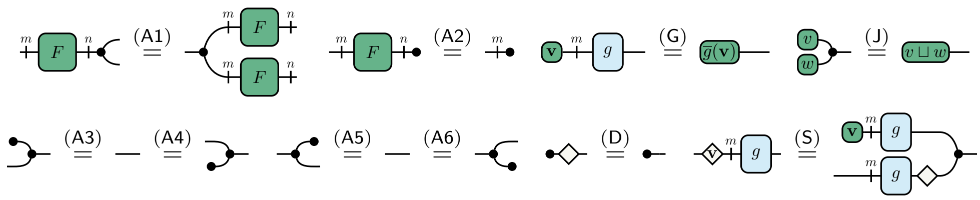
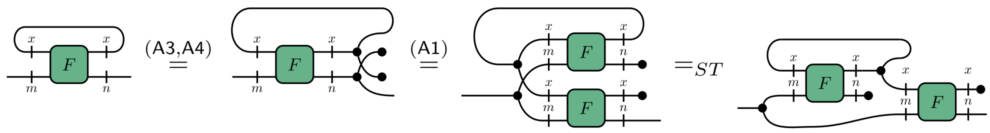


Figure 6: Sets $\mathcal{A} := \{\text{A1}, \text{A2}, \text{A3}, \text{A5}, \text{A5}, \text{A6}\}$, $\mathcal{C} := \{\text{G}, \text{J}\}$ and $\mathcal{D} := \{\text{D}, \text{S}\}$.

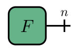
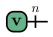
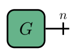
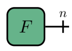
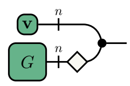
Example 57 (Unfolding)**.** The *unfolding* rule is an important equation that holds in any dataflow category [Has97]. It is derived equationally by using equations from $\mathcal{A}$.

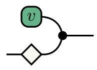


In [GJL17a], the unfolding rule is crucial, and it will be used in the productivity result below.

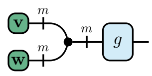
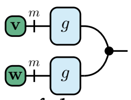
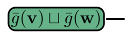
5.2 Productivity

A common use of equational reasoning is to take a circuit, precompose it with some values, and reduce it to its stream of output values. Circuits which can be reduced in this way are called *productive*. A circuit is *closed* if it is a morphism $0 \to n$ and open otherwise.

Definition 58 (Productivity)**.** *For a set of equations $\mathcal{E}$, a closed sequential circuit*  *is called* productive *under $\mathcal{E}$ if there exist values*  *and a closed sequential circuit*  *such that*  $=_{\mathcal{E}}$ *. The circuit on the right is in* productive form.

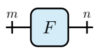
The join of a value and a delay as seen above is common: it can be thought of as a register with an initial value. For brevity, it will often be written as  $:=$ .

Core equations for reducing circuits in any interpretation are listed in the right of Fig. 6, adapted from those in [GJ16]. Equation (G) is used to apply a gate to some values; equation (J) expresses how a join coalesces two values using the join operation $\sqcup$ in the lattice $\mathbf{V}$; and (D) enforces that the 'initial value' of a delay is the disconnected $\perp$ value.

Equation (S) is perhaps the most unexpected and it requires some explanation. Intuitively, it says that the join generator is 'almost' a natural transformation. In general this is not the case:  is not the same as  as the former reduces to  while the latter reduces to . However, when one of the inputs is guarded by a delay then there is no need to combine the inputs, so a guarded form of naturality holds.

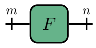
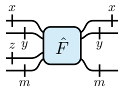
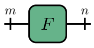
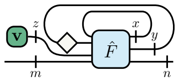
Remark 59. Equation (S) models *retiming* [LS91]: moving registers forward or backward across gates. Forward retiming (the LHS of (S)) is always possible but for backward retiming (the RHS of (S)), the value in the register must be in the image of the gates.

Proposition 60 (Extensionality)**.** *Any combinational circuit is productive under $\mathcal{C} + \mathcal{A}$.*

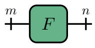
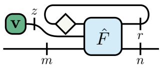
Proof. By induction over the structure of . □

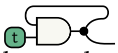
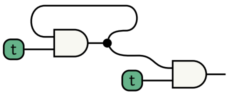
5.3 Non-delay-guarded feedback

Unfortunately, the framework presented in [GJ16] was not *complete*: the equations could not necessarily handle circuits with *non-delay-guarded feedback*, in which some feedback loops do not pass through a delay generator. A circuit is called *passive* if it contains no values.

Lemma 61 (Trace-delay form)**.** *For any sequential circuit*  *, there exists a passive combinational*  *and values*  *such that*  $=_{ST}$ 

Proof. Any trace can become a 'global trace' by applying *tightening* and *superposing*. For the delays, *yanking* is used to create a feedback loop, and *sliding* is used to shift delays around the trace. □

Definition 62. *A sequential circuit*  *has* delay-guarded feedback *if its global trace-delay form is*  *i.e. all feedback passes through a delay.*

Example 63. Consider the circuit . In the stream semantics, this produces the constant $\perp$ stream, but this cannot be obtained by applying equations in $\mathcal{C} + \mathcal{A}$. The only option is to unfold which results in . Once again, the only option is to unfold: we are stuck.

One might ask if the delay-guarded feedback condition should be enforced in order to assert productivity. However, careful use of non-delay-guarded feedback can still result in productive circuits: for example, it can be used as a clever way of sharing resources [Rie04].

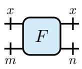
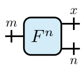
Instead, an equation is required to transform 'instant feedback' into something more workable. Inspiration can be gleaned from the following:

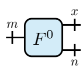
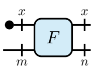
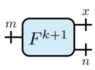
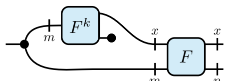
Theorem 64 (Kleene fixed-point theorem ([SLG94], pp. 24))**.** *Every Scott continuous function $f : \mathbf{V} \to \mathbf{V}$ on a lattice $\mathbf{V}$ has a least fixed point in $\mathbf{V}$: the supremum of $\{f^n(\perp) \mid n \in \mathbb{N}\}$.*

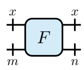
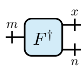
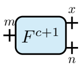
Lemma 65. *For a monotone function $f : \mathbf{V}^{n+m} \to \mathbf{V}^n$ and $i \in \mathbb{N}$, let $f_i : \mathbf{V}^m \to \mathbf{V}^n$ be defined as $f_0(x) = f(\perp, x)$ and $f_{k+1}(x) = f(f_k(x), x)$. Let c be the length of the longest chain in the value lattice $\mathbf{V}^n$. Then, for $j > c$, $f_c(x) = f_j(x)$.*

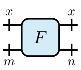
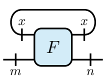
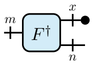
Proof. Since f is monotone, it has a least fixed point by the Kleene fixed-point theorem. This will either be some value v or, since $\mathbf{V}$ is finite, the $\top$ element. The most iterations of f it would take to obtain this fixpoint is c, i.e. the function produces a value one step up the lattice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

This suggests a new family of equations.

Definition 66. *For a combinational circuit*  *, let its n*th iteration  *be defined inductively as*

 $:=$  *and*  $:=$  .

Definition 67. *For an interpretation $\mathcal{I} = (\mathbf{V}, \mathcal{I}_{\mathcal{V}}, \mathcal{I}_{\mathcal{G}})$, the* fixpoint *of*  *, denoted as*  *, is defined as*  *where c is the length of the longest chain in $\mathbf{V}$.*

Proposition 68. *For any combinational circuit*  *,*  $\approx_{\mathcal{I}}$  (IF).

Proof. By applying Lemma 65 pointwise. □

If applied locally for every feedback loop, the (IF) equation would cause an exponential blowup of the circuit. However, if a circuit is in global trace-delay form, the (IF) equation need only be applied once to the 'global feedback loop'.

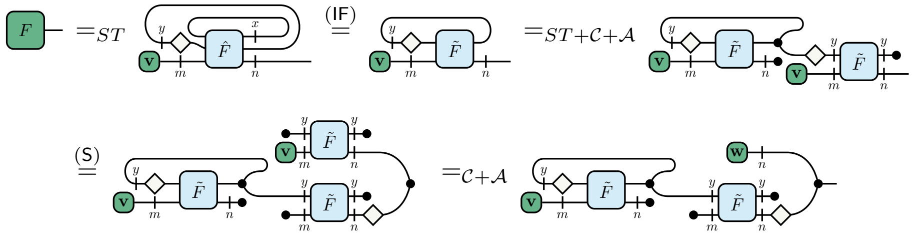
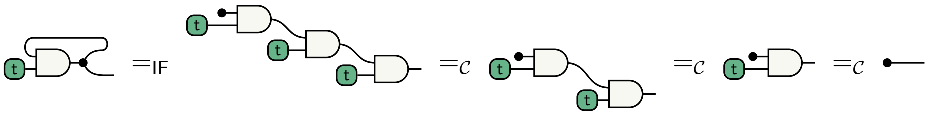


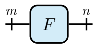
Figure 7: Proof of Theorem 71.

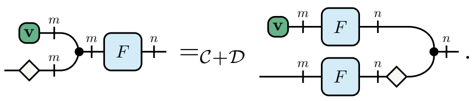
Example 69. Recall the circuit in Example 63. Using (IF), the circuit *does* reduce to a value:

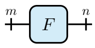


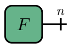
Using (IF), combinational feedback can be exhaustively unfolded. However, *delay-guarded* feedback cannot be treated this way if the circuit has open inputs. In stream functions corresponding to combinational circuits, the ith element of the output stream is computed solely using the ith input by definition. The trace of a function in $\mathbf{Stream}_{\mathcal{I}}$ is computed using the least fixpoint as defined in Proposition 15; as morphisms in $\mathbf{Stream}_{\mathcal{I}}$ are monotone and the lattice $\mathbf{V}$ is finite, this can be computed in finite iterations. Moreover, since the trace is applied pointwise, the traced function is also a function in which the ith output depends only on the ith input, so it can be expressed as a syntactic combinational circuit.

In the delay-guarded case, the stream function is constructed using the *shift* function, so the $(i+1)$th output will depend on at least the ith input. Without knowing the inputs in advance, it is not possible to construct a syntactic circuit without feedback that is equal to a delay-guarded circuit. If a circuit *does* have no inputs, a productivity result can be derived.

Lemma 70 (Generalised streaming)**.** *For any combinational circuit*  *,*

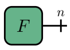
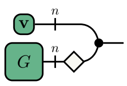
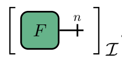


Proof. By induction over the structure of  . □

Theorem 71. *Any closed sequential circuit*  *is productive under* $\mathcal{C}+\mathcal{D}+\mathcal{A}+\mathsf{IF}$.

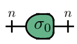
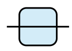
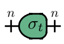
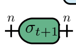
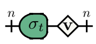
Proof. Shown in Fig. 7. First the circuit is put into global trace-delay by Lemma 61. Then IF is applied to create a circuit with only delay-guarded feedback, which is then unfolded. By using the fact that  is the unit of the join, *streaming* can be repeatedly applied. Since the 'top' copy is a closed combinational circuit, it can be reduced to some values by Proposition 60. □

Although an instant value will always be obtained, it may not be 'useful': it could be the $\perp$ value as in Example 69. This can be thought of as a default value for unbounded recursion.

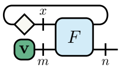
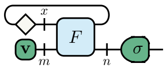
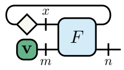
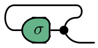
Lemma 72. *Given a sequential circuit*  *with productive form* *, let f be its stream* *. Then, $f(\bullet)(0) = \mathbf{v}$.*

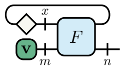
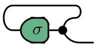
Proof. The value  is the only component not guarded by a delay. □

Using productivity, the elements of the corresponding stream are enumerated. However, it is desirable to transform a circuit into *just* its stream of output values.

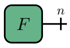
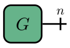
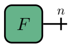
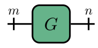
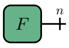
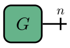
Definition 73 (Waveform)**.** *The empty waveform is defined as*  := *. For a given head value*  *and waveform*  *of length t, a waveform of length $t+1$ is defined as*  :=  *.*

If, when applying productivity, the state of the delayed subcircuit reoccurs, then the produced values must also reoccur. This is expressed in the following equation: we write $\mathcal{R}$ for $\mathcal{C} + \mathcal{D} + \mathsf{IF}$.

Proposition 74. *If*  $=_{\mathcal{R}+\mathcal{A}}$ , *then*  $\approx_{\mathcal{I}}$  (P).

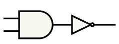
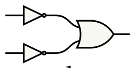
Proof. If a circuit is equal to itself followed by a waveform, then by applying the same sequence of equations repeatedly, multiple copies of the waveform can be created. Since the equations in $\mathcal{R} + \mathcal{A}$ are sound, no other values can be obtained: by Lemma 72, $\left[\!\left[\right.\right.$  $\left.\left.\right]\!\right]_{\mathcal{I}} = \sigma :: \sigma :: \sigma :: \cdots$. This is the interpretation of the circuit . □

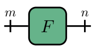
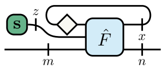
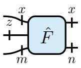
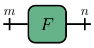
By combining Theorem 71 and (P), any two closed circuits with the same stream are equal to the same waveform in the equational theory.

Theorem 75. *For two closed sequential circuits*  *and* ,  $=_{\mathcal{R}+\mathcal{A}+\mathsf{P}}$  *if and only if* $\left[\right.$  $\left.\right]_{\mathcal{I}} = \left[\right.$  $\left.\right]_{\mathcal{I}}$.

Proof. The $(\Leftarrow)$ direction is immediate. The $(\Rightarrow)$ direction follows by applying Theorem 71 repeatedly: since the ith value obtained corresponds to the ith value of the stream by Lemma 72 and any stream function in $\mathbf{Stream}_{\mathcal{I}}$ has finitely many stream derivatives, eventually a periodic segment will be reached. Since the productivity procedure is deterministic, the internal state of the circuit will also reoccur. Finally, (P) is applied to transform each circuit into the same infinite waveform structure. □

5.4 Full abstraction

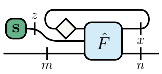
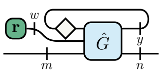
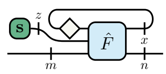
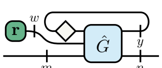
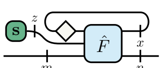
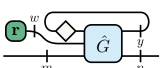
In the closed case these equations suffice as the input values are propagated across the circuit, with gates evaluated one by one. However, for an *open circuit* this is not enough. For example, consider the circuits  and : when interpreted under $\mathcal{I}_\star$ their stream functions are equal, but the equations so far cannot translate one to the other.

A general equation must be defined to translate between open circuits with the same input-output behaviour. A circuit  is in *global delay form* if it is in the form . We call the circuit  a *combinational core* of . Unfortunately, there is no guarantee that the combinational cores of extensionally equivalent circuits will model the same function. Instead, we consider the *bisimilar states* in the underlying Mealy machine: we tweak the standard definition of bisimilarity for Mealy machines [BRS08].

Definition 76 (Bisimilarity)**.** *Given two $\mathcal{I}$-Mealy machines A and B with state sets $\mathbf{V}^x$ and $\mathbf{V}^y$ respectively, $(s,t) \in \mathbf{V}^x \times \mathbf{V}^y$ are* bisimilar states *if $s[a] = t[a]$ and (s_a, t_a) are bisimilar states. A and B are* bisimilar *if their initial states $\bar{s}^A$ and $\bar{s}^B$ are bisimilar states.*

Lemma 77. *Let A and B be two $\mathcal{I}$-Mealy machines with initial states $\bar{s}^A$ and $\bar{s}^B$. Then A and B are bisimilar if and only if $!A = !B$.*

Proof. For the $(\Rightarrow)$ direction, the streams are computed as $\bar{s}^A[v_0] :: \bar{s}^A_{v_1}[v_0] :: \cdots$. If the initial states are bisimilar, then the output of the Mealy machine after some number of transitions must be equal, so the streams themselves are equal. For the $(\Leftarrow)$ direction, assume that $!A = !B$ but A and B are not bisimilar. Then there exists some number of transitions i after which the output of A and B are not equal. But this would mean that $!A(i) \neq !B(i)$, and therefore $!A \neq !B$. Therefore if $!A = !B$ then A and B must be bisimilar. □

Corollary 78. *Let*  *and*  *be two sequential circuits such that*  $\approx_{\mathcal{I}}$ . *Then* $\left[\!\left[\right.\right.$  $\left.\left.\right]\!\right]_{\mathcal{I}}$ *and* $\left[\!\left[\right.\right.$  $\left.\left.\right]\!\right]_{\mathcal{I}}$ *are bisimilar.*

Lemma 79. *Given a sequential circuit in global delay form* *and* $M :=$ $\llbracket \; \rrbracket_{\mathcal{I}} =$ $(\mathbf{V}^{x+z}, \langle s[a], s_a \rangle)$, $\downarrow \left(\; \right) = \mathsf{s}' +\!\!+ \mathsf{s}_\mathsf{v}$ *and* $\ulcorner\downarrow\left(\; \right)\urcorner +\!\!+ \perp^z = \mathsf{s}' +\!\!+ \mathsf{s}_\mathsf{v}$.

Proof. The transition function is computed for state $a +\!\!+ b$ and inputs c as

$$
\begin{aligned}
(a +\!\!+ b)_c &= (\mathsf{fix}_{(a +\!\!+ b),c}, \perp^z) \\
&= (\perp^x, \perp^z) \sqcup (\pi_x(F(a,b,c)), \perp^z) \sqcup (\pi_x(F(a,b,c)), \perp^z) \sqcup \cdots \\
&= (\pi_x(F(a,b,c)), \perp^z)
\end{aligned}
$$

and the ouput function as $(a +\!\!+ b)[c] = F(a,b,c)$. This corresponds to the syntactic equations. □

The notion of bisimilar states can be mimicked syntactically.

Notation 80. *Given a circuit* *with* *combinational, let* *be values such that* $=_{\mathcal{C}+\mathcal{A}}$. *We write* $\downarrow\left(\; \right) :=$.

Definition 81 (Transitions)**.** *Let* *and* *be combinational cores. Then, let*

$$\mathsf{T}(-) := \left(\; , \; , \; , \; \right) \mapsto \left\{ \left(\downarrow \left(\; \right), \downarrow \left(\; \right) \right) \;\middle|\; \mathbf{v} \in \mathbf{V}^m \right\}.$$

Definition 82 (State relation)**.** *For two circuits* *and* *, let the initial transition* $\mathsf{T}_0 := \mathsf{T}(\; , \; , \; , \;)$ *and let their* state relation *be the smallest set with* T_0 *as a subset which is closed under* $(\; , \;) \mapsto \mathsf{T}(\; , \; , \; , \;)$.

Lemma 83. *The state relation for any two circuits is finite.*

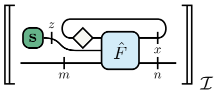
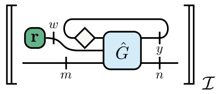
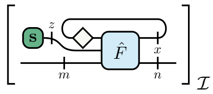
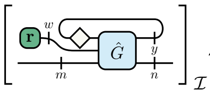
Proof. The value set $\mathbf{V}^x \times \mathbf{V}^y$ is finite. □

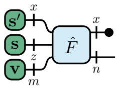
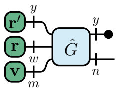
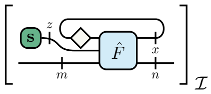
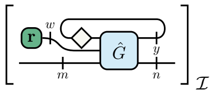
Lemma 84. *Let* *and* *be two sequential circuits such that* $\approx_{\mathcal{I}}$ *, and let M and M' be their corresponding Mealy machines. Then,* $(\; , \;)$ *is a member of the state relation of* *and* *if and only if* $(\mathbf{s}, \mathbf{r})$ *are bisimilar states in M and M'.*

Proof. M and M' are bisimilar by Corollary 78. Therefore $(\perp^x +\!\!+ \mathbf{s}, \perp^y +\!\!+ \mathbf{r})$ are bisimilar states; the corresponding tuple is contained in the state relation by definition. The set of bisimilar states is generated by repeatedly applying the transition function for all inputs. By Lemma 79, the function that generates the state relation corresponds to the transition function. Therefore each pair in the state relation corresponds to a pair of bisimilar states. □

This can be used to formulate a new family of equations.

Proposition 85 (Mealy equation)**.** *Given two sequential circuits* *and* *with global delay forms* $=_{\mathcal{ST}+\mathcal{R}}$ *and* $=_{\mathcal{ST}+\mathcal{R}}$ *, let B be their state relation. If, for each* $\left(\; , \; \right) \in B$ *and* $\in \mathbf{V}^m$, $=_{\mathcal{C}}$ *and* $=_{\mathcal{C}}$ *, then* $\approx_{\mathcal{I}}$ (MM).

Proof. Let $M :=$  and $M' :=$ . For  to be equal to , M and M' must be bisimilar by Lemma 77 and Theorem 29.

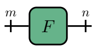
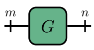
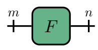
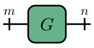
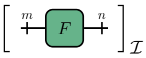
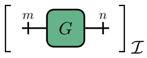
By definition, M and M' are bisimilar if $s[v]^M = r[v]$ when s and r are bisimilar states. By Lemma 84, the pairs in the state relation correspond precisely to the bisimilar states in M and M'. Therefore, if we check that  $=_{\mathcal{C}}$  for all pairs in the state relation, we have confirmed that the outputs of each pair of bisimilar states are equal, and thus  $=$ . □

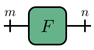
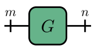
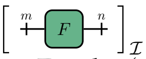
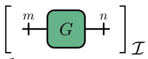
While the (MM) equation may look obtuse at first glance, more familiar equivalences can be derived using it. In the case of $\mathcal{I}_\star$, a set of useful derivable equations is listed in Fig. 8.

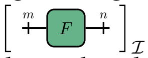
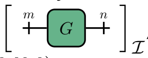
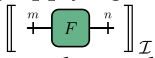
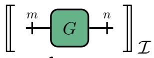
Lemma 86. *The equations in $\mathcal{D}$ and $\mathcal{A}$ are derivable using* (MM)*.*

Proof. Since the equations are open and both sides have the same semantics, it is a simple exercise to derive them using (MM). □

We now have enough to achieve *full abstraction*: a correspondence between the denotational model and the equational model.

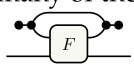
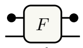
Theorem 87 (Full abstraction)**.** *For two sequential circuits*  *and* *,*  $=_{\mathcal{C}+\mathsf{MM}}$  *if and only if*  $=$ *.*

Proof. The ($\Rightarrow$) direction follows as  $=_{\mathcal{R}+\mathsf{MM}}$  as all equations are valid if and only if  $=$ .

For the ($\Leftarrow$) direction, any circuit is brought into global delay form by applying Lemma 61 followed by the instant feedback equation (IF). Since  $=$ ,  and  are bisimilar by Lemma 77. Since they are bisimilar, the (MM) equation can be used to transform one circuit into the other, completing the proof. □

6 Related and further work

Algebraic (categorical) and diagrammatic semantics for combinational (i.e. no feedback and no delays) Boolean circuits were first given in [Laf03]. By lifting the set of values to a lattice it was possible to extend this framework with delay and feedback, allowing the axiomatisation of sequential circuits [GJ16; GJL17a]. However, in *loc. cit.* interpretations, equations and quotients were layered, resulting in a presentation that emphasised certain methodological points at the cost of mathematical clarity and organisation. The new presentation in this paper is more direct: the syntax and semantics are neatly separated, with the latter formally defined as a prop morphism into stream functions.

String diagrams as a graphical syntax for monoidal categories was introduced a few decades ago [JS91; JSV96], but more recently we have witnessed an explosion in their use for various applications, such as quantum protocols [AC04], signal flow diagrams [BSZ14; BSZ15], linear algebra [BSZ17; Zan15; Bon+19], and dynamical systems [BE15; FSR16]. While these frameworks use compositional circuits in some way, the nature of digital circuits mean there are some differences in our system. In many of the above applications, the join and the fork form a *Frobenius structure*, so a trace is constructed as . Since our setting is *Cartesian*, such a construction degenerates to .

There are other settings that permit loops but retain unidirectionality of wires. *Categories with feedback* were introduced in [KSW02] as a weakening of STMCs that removes the yanking axiom, enforcing that *all* traces are delay-guarded. In [Di +21] Mealy machines are characterised as a category with feedback: this is compatible with our framework since all 'instant feedback' is expressed as fixpoints and

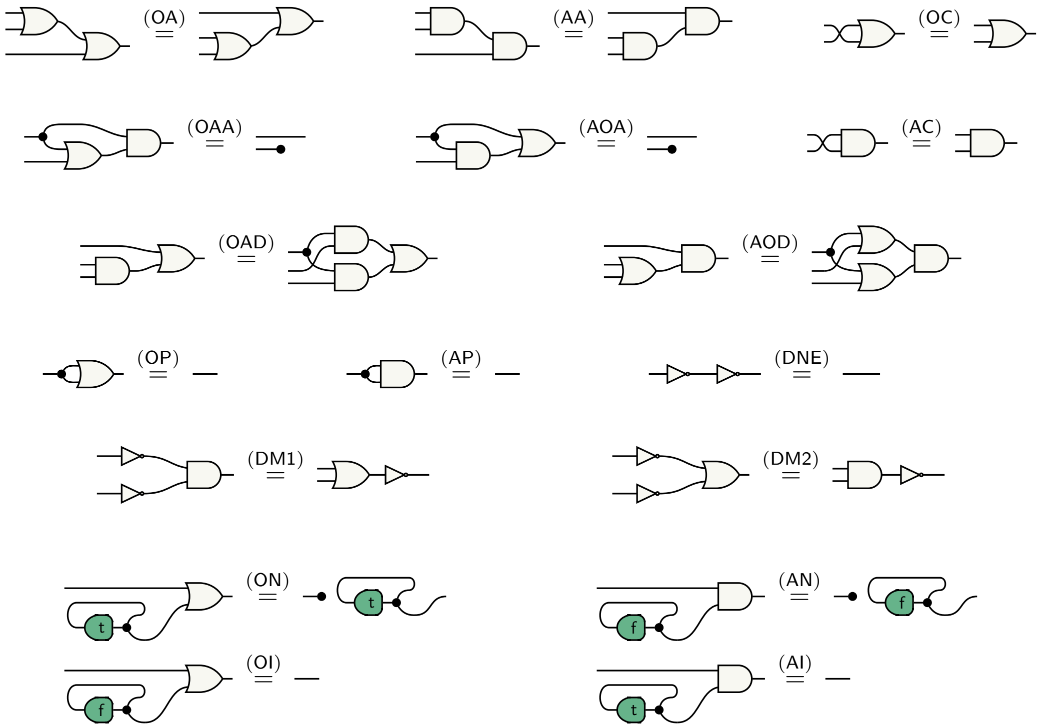


Figure 8: Some common equations that derivable for $\mathcal{I}_\star$ using (MM)

only delay-guarded feedback remains. *Categories with delayed trace* [SK19] weaken the notion further by removing the sliding axiom; this prohibits the unfolding rule so would be unsuitable.

Axiomatising fixpoint operators has been studied extensively [BÉ93; Ste00; SP00]. Since any Cartesian traced category admits a fixpoint (or *Conway*) operator [Has97], these equations can be expressed using the Cartesian naturality equations $\mathcal{A}$ and the axioms of STMCs. However, since our work takes place in a *finite* lattice, we are able to define a new effective equation, in which a fixpoint can be expressed by iterating the circuit a finite number of times. While this result is well-known from the denotational perspective [SLG94], it has not been used before to solve the problem of combinational feedback. We can only speculate that perhaps the reason why this proved elusive is because the non-compositional or non-diagrammatic formulation of circuits made its applicability less obvious. The interplay of causal streams and dataflow categories has also been studied elsewhere: recently, a generalisation of causal streams known as *monoidal streams* [DS22] has been developed to provided semantics to dataflow programming. Although this generalises some aspects of the streams used in our work, our approach differs in the use of the finite lattice and exclusively monotone functions.

The correspondence between Mealy machines and digital circuits is a fundamental result in automata theory [Mea55] applied extensively in circuit design [KJ09]. The links between Mealy machines and causal stream functions using coalgebras is a more recent development [Rut05a; Rut05b; Rut06]. Mealy machines over meet-semilattices are introduced in [BRS08] to model a logical framework which includes fixpoint. We also employ this technique in order to deal with fixpoint, but our work adds more structure in that we show the $\mathcal{I}$-Mealy machines form a prop. By enforcing a monotone Mealy function and concretely assigning each state a power of values, the link between circuits and Mealy machines is expressible as a prop morphism.

Reasoning with string diagrams is not an efficient syntax to work with computationally. For an efficient operational semantics, string diagrams must be translated to combinatorial graphs: this was touched on informally in [GJL17a], which used *framed point graphs* [Kis12]. Recent work in string diagram rewriting [Bon+22a; Bon+21; Bon+22b] has used *hypergraphs* to perform rewriting modulo Frobenius structure. This framework has been adapted for traced categories [Kay21] and categories with a comonoid structure [FL22]. Since these are the two main structures at play in our setting, the two frameworks could be combined for rewriting sequential circuits.

References

[AC04] Samson Abramsky and Bob Coecke. "A Categorical Semantics of Quantum Protocols". In: *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.* July 2004, pp. 415–425. DOI: 10.1109/LICS.2004.1319636.

[BE15] John C. Baez and Jason Erbele. "Categories in Control". In: *Theory and Applications of Categories* 30.24 (May 20, 2015), pp. 836–881. DOI: 10.48550/arXiv.1405.6881.

[BÉ93] Stephen L. Bloom and Zoltán Ésik. "Iteration Theories". In: *Iteration Theories: The Equational Logic of Iterative Processes*. Ed. by Stephen L. Bloom and Zoltán Ésik. EATCS Monographs on Theoretical Computer Science. Berlin, Heidelberg: Springer, 1993, pp. 159–213. ISBN: 978-3-642-78034-9. DOI: 10.1007/978-3-642-78034-9_7.

[Bel77] Nuel D. Belnap. "A Useful Four-Valued Logic". In: *Modern Uses of Multiple-Valued Logic*. Ed. by J. Michael Dunn and George Epstein. Episteme. Dordrecht: Springer Netherlands, 1977, pp. 5–37. ISBN: 978-94-010-1161-7. DOI: 10.1007/978-94-010-1161-7_2.

[Bon+19] Filippo Bonchi et al. "Graphical Affine Algebra". In: *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. June 2019, pp. 1–12. DOI: 10.1109/LICS.2019.8785877.

[Bon+21] Filippo Bonchi et al. *String Diagram Rewrite Theory II: Rewriting with Symmetric Monoidal Structure*. Apr. 29, 2021. DOI: 10.48550/arXiv.2104.14686. arXiv: 2104.14686 [cs, math].

[Bon+22a] Filippo Bonchi et al. "String Diagram Rewrite Theory I: Rewriting with Frobenius Structure". In: *Journal of the ACM* 69.2 (Mar. 10, 2022), 14:1–14:58. ISSN: 0004-5411. DOI: 10.1145/3502719.

[Bon+22b] Filippo Bonchi et al. "String Diagram Rewrite Theory III: Confluence with and without Frobenius". In: *Mathematical Structures in Computer Science* (June 13, 2022), pp. 1–41. ISSN: 0960-1295, 1469-8072. DOI: 10.1017/S0960129522000123.

[BRS08] M. M. Bonsangue, Jan Rutten, and Alexandra Silva. "Coalgebraic Logic and Synthesis of Mealy Machines". In: *Foundations of Software Science and Computational Structures*. Ed. by Roberto Amadio. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 231–245. ISBN: 978-3-540-78499-9. DOI: 10.1007/978-3-540-78499-9_17.

[BSZ14] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. "A Categorical Semantics of Signal Flow Graphs". In: *CONCUR 2014 – Concurrency Theory*. Ed. by Paolo Baldan and Daniele Gorla. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2014, pp. 435–450. ISBN: 978-3-662-44584-6. DOI: 10.1007/978-3-662-44584-6_30.

[BSZ15] Filippo Bonchi, Pawel Sobociński, and Fabio Zanasi. "Full Abstraction for Signal Flow Graphs". In: *ACM SIGPLAN Notices* 50.1 (Jan. 14, 2015), pp. 515–526. ISSN: 0362-1340. DOI: 10.1145/2775051.2676993.

[BSZ17] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. "Interacting Hopf Algebras". In: *Journal of Pure and Applied Algebra* 221.1 (Jan. 1, 2017), pp. 144–184. ISSN: 0022-4049. DOI: 10.1016/j.jpaa.2016.06.002.

[CŞ90] Virgil Emil Căzănescu and Gheorghe Ştefănescu. "Towards a New Algebraic Foundation of Flowchart Scheme Theory". In: *Fundamenta Informaticae* 13.2 (Jan. 1, 1990), pp. 171–210. ISSN: 0169-2968. DOI: 10.3233/FI-1990-13204.

[Di +21] Elena Di Lavore et al. "A Canonical Algebra of Open Transition Systems". In: *Formal Aspects of Component Software*. Ed. by Gwen Salaün and Anton Wijs. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 63–81. ISBN: 978-3-030-90636-8. DOI: 10.1007/978-3-030-90636-8_4.

[DS22] Elena Di Lavore and Paweł Sobociński. "Monoidal Width". Feb. 15, 2022. arXiv: 2202.07582 [cs, math].

[FL22] Tobias Fritz and Wendong Liang. *Free Gs-Monoidal Categories and Free Markov Categories*. Apr. 17, 2022. DOI: 10.48550/arXiv.2204.02284.

[FS18] Brendan Fong and David I. Spivak. *Seven Sketches in Compositionality: An Invitation to Applied Category Theory*. Oct. 12, 2018. DOI: 10.48550/arXiv.1803.05316. arXiv: 1803.05316 [math].

[FSR16] Brendan Fong, Paweł Sobociński, and Paolo Rapisarda. "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. New York, NY, USA: Association for Computing Machinery, July 5, 2016, pp. 495–504. ISBN: 978-1-4503-4391-6. DOI: 10.1145/2933575.2934556.

[GJ16] Dan R. Ghica and Achim Jung. "Categorical Semantics of Digital Circuits". In: *2016 Formal Methods in Computer-Aided Design (FMCAD)*. Oct. 2016, pp. 41–48. DOI: 10.1109/FMCAD.2016.7886659.

[GJL17a] Dan R. Ghica, Achim Jung, and Aliaume Lopez. "Diagrammatic Semantics for Digital Circuits". In: *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*. Ed. by Valentin Goranko and Mads Dam. Vol. 82. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, 24:1–24:16. ISBN: 978-3-95977-045-3. DOI: 10.4230/LIPIcs.CSL.2017.24.

[GJL17b] Dan R. Ghica, Achim Jung, and Aliaume Lopez. "Diagrammatic Semantics for Digital Circuits (Technical Report)". In: *CoRR* abs/1703.10247 (Mar. 29, 2017). arXiv: 1703.10247.

[Has09] Masahito Hasegawa. "On Traced Monoidal Closed Categories". In: *Mathematical Structures in Computer Science* 19.2 (Apr. 2009), pp. 217–244. ISSN: 1469-8072, 0960-1295. DOI: 10.1017/S0960129508007184.

[Has97] Masahito Hasegawa. "Recursion from Cyclic Sharing: Traced Monoidal Categories and Models of Cyclic Lambda Calculi". In: *Typed Lambda Calculi and Applications*. Ed. by Philippe de Groote and J. Roger Hindley. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1997, pp. 196–213. ISBN: 978-3-540-68438-1. DOI: 10.1007/3-540-62688-3_37.

[JS91] André Joyal and Ross Street. "The Geometry of Tensor Calculus, I". In: *Advances in Mathematics* 88.1 (1991), pp. 55–112. ISSN: 0001-8708. DOI: 10.1016/0001-8708(91)90003-P.

[JSV96] André Joyal, Ross Street, and Dominic Verity. "Traced Monoidal Categories". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 119.3 (Apr. 1996), pp. 447–468. ISSN: 1469-8064, 0305-0041. DOI: 10.1017/S0305004100074338.

[Kay21] George Kaye. *Rewriting Graphically with Symmetric Traced Monoidal Categories*. Mar. 18, 2021. DOI: 10.48550/arXiv.2010.06319. arXiv: 2010.06319 [math].

[Kis12] Aleks Kissinger. "Pictures of Processes: Automated Graph Rewriting for Monoidal Categories and Applications to Quantum Computing". PhD thesis. University of Oxford, Mar. 22, 2012. DOI: 10.48550/arXiv.1203.0202. arXiv: 1203.0202 [quant-ph].

[KJ09] Zvi Kohavi and Niraj K. Jha. *Switching and Finite Automata Theory*. Cambridge University Press, Oct. 22, 2009. 630 pp. ISBN: 978-1-139-48308-7. DOI: 10.1017/CBO9780511816239.

[KSW02] P. Katis, Nicoletta Sabadini, and Robert F. C. Walters. "Feedback, Trace and Fixed-Point Semantics". In: *RAIRO - Theoretical Informatics and Applications* 36.2 (Apr. 2002), pp. 181–194. ISSN: 0988-3754, 1290-385X. DOI: 10.1051/ita:2002009.

[Lac04] Stephen Lack. "Composing PROPs". In: *Theory and Applications of Categories* 13.9 (2004), pp. 147–163. ISSN: 1201-561X.

[Laf03] Yves Lafont. "Towards an Algebraic Theory of Boolean Circuits". In: *Journal of Pure and Applied Algebra* 184.2 (Nov. 1, 2003), pp. 257–310. ISSN: 0022-4049. DOI: 10.1016/S0022-4049(03)00069-0.

[LS91] Charles E. Leiserson and James B. Saxe. "Retiming Synchronous Circuitry". In: *Algorithmica* 6.1 (June 1, 1991), pp. 5–35. ISSN: 1432-0541. DOI: 10.1007/BF01759032.

[Mea55] George H. Mealy. "A Method for Synthesizing Sequential Circuits". In: *The Bell System Technical Journal* 34.5 (Sept. 1955), pp. 1045–1079. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1955.tb03788.x.

[MSB12] Michael Mendler, Thomas R. Shiple, and Gérard Berry. "Constructive Boolean Circuits and the Exactness of Timed Ternary Simulation". In: *Formal methods in system design : an international journal* 40.3 (2012), pp. 283–329. ISSN: 0925-9856. DOI: 10.1007/s10703-012-0144-6.

[Rie04] Marc D. Riedel. "Cyclic Combinational Circuits". PhD thesis. United States: California Institute of Technology, May 27, 2004. 112 pp. ISBN: 9780496071005.

[Rut05a] J. J. M. M. Rutten. “A Coinductive Calculus of Streams”. In: *Mathematical Structures in Computer Science* 15.1 (Feb. 2005), pp. 93–147. ISSN: 1469-8072, 0960-1295. DOI: 10.1017/S0960129504004517.

[Rut05b] J. J. M. M. Rutten. “Algebra, Bitstreams, and Circuits”. In: *Software Engineering* [*SEN*] R 0502 (Jan. 1, 2005).

[Rut06] J. J. M. M. Rutten. “Algebraic Specification and Coalgebraic Synthesis of Mealy Automata”. In: *Electronic Notes in Theoretical Computer Science*. Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005) 160 (Aug. 8, 2006), pp. 305–319. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2006.05.030.

[Sel11] Peter Selinger. “A Survey of Graphical Languages for Monoidal Categories”. In: *New Structures for Physics*. Ed. by Bob Coecke. Lecture Notes in Physics. Berlin, Heidelberg: Springer, 2011, pp. 289–355. ISBN: 978-3-642-12821-9. DOI: 10.1007/978-3-642-12821-9_4.

[SK19] David Sprunger and Shin-ya Katsumata. “Differentiable Causal Computations via Delayed Trace”. In: *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science* (*LICS*). June 2019, pp. 1–12. DOI: 10.1109/LICS.2019.8785670.

[SLG94] V. Stoltenberg-Hansen, I. Lindström, and E. R. Griffor. *Mathematical Theory of Domains*. Cambridge University Press, Sept. 22, 1994. 366 pp. ISBN: 978-0-521-38344-8. DOI: 10.1017/CBO9781139166386.

[SP00] A. Simpson and G. Plotkin. “Complete Axioms for Categorical Fixed-Point Operators”. In: *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science* (*Cat. No.99CB36332*). June 2000, pp. 30–41. DOI: 10.1109/LICS.2000.855753.

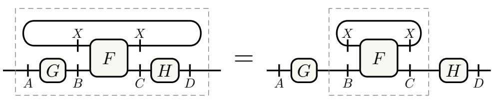
[Ste00] Gheorghe Stefanescu. *Network Algebra*. Discrete Mathematics and Theoretical Computer Science. Springer Science & Business Media, Apr. 12, 2000. 422 pp. ISBN: 978-1-85233-195-5. DOI: 10.1007/978-1-4471-0479-7.

[Zan15] Fabio Zanasi. “Interacting Hopf Algebras: The Theory of Linear Systems”. PhD thesis. University of Lyon, Oct. 5, 2015. DOI: 10.48550/arXiv.1805.03032. arXiv:1805.03032.

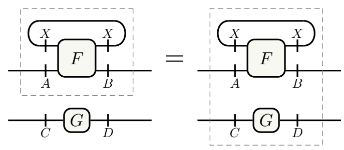
A Axioms of symmetric traced monoidal categories

These axioms were originally presented in [JSV96].

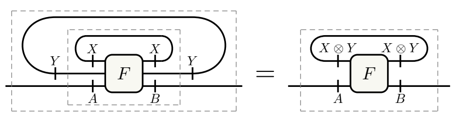
Tightening



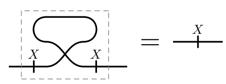
Superposing



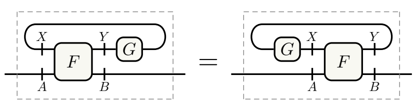
Vanishing



Yanking



Sliding



An additional vanishing axiom appeared in [JSV96]; it is shown in [Has09] that this was redundant.