A compositional theory of digital circuits

Dan R. Ghica
School of Computer Science
University of Birmingham, UK
d.r.ghica@bham.ac.uk

George Kaye
School of Computer Science
University of Birmingham, UK
g.j.kaye@pgr.bham.ac.uk

David Sprunger
Department of Mathematics & Computer Science
Indiana State University, USA
david.sprunger@indstate.edu

Abstract

A syntax 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 having been 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 (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 using an equational theory. The goals of this paper are twofold. First we formalise the semantics of digital circuits specified compositionally by interpreting them as functions on streams with certain properties. Second we refine the previous equational theory so that it is in perfect agreement with the semantic model. To support this result we introduce two key equations: the first can eliminate non-delay-guarded feedback via finite unfoldings, and the second can translate between circuits with the same behaviour syntactically by reducing the problem to checking a finite number of closed circuits. While these are enough to establish a correspondence between the denotational and the equational frameworks, we also show how simpler equations can be derived for more intuitive reasoning. The most important consequence of this 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 built from 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. While a compositional syntax for *combinational* circuits, which model functions arises naturally [Laf03], the case of *sequential* circuits, which contain delay and feedback, is more subtle. However, fully compositional syntactic and categorical accounts of circuits have recently led to novel operational, rewriting-based, semantics [GJ16; GJL17a].

In the current paper we bring together these two semantic models, denotational and categorical, to give a fully compositional interpretation of digital circuits including those which may have non-delay-guarded feedback loops. To be more precise, by 'digital circuits' we primarily understand *electronic* 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), nor *electrical* circuits of resistors and capacitors, which are quite different [BS22].

In previous work, the semantics of circuits were presented informally; in this paper these semantics are formalised using a morphism from the category of (syntactic) digital circuits to a category of *stream functions*. Streams are infinite sequences of digital values which are the input and output of digital circuits; a stream function consumes and produces streams. However, stream functions that are representations of circuits have certain characteristics. They are *causal* and *monotone*, properties related to hardware. Moreover, they specify finitely many possible behaviours, a characteristic of the finite-state nature of digital circuits. The first technical result of this paper is to to show that the every circuit constructed compositionally corresponds to one of these stream functions, and conversely every such stream function has a corresponding set of syntactic circuits with that stream function as its behaviour. Along the way, we also lift the well-known formalism of Mealy machines [Mea55] to work with lattices: as stream functions are themselves a form of Mealy machine [Rut06], this is an essential step in showing the correspondence between sequential circuits and stream functions.

The equational theory of digital circuits is a symmetric traced Cartesian category (or 'dataflow category') augmented with special domain-specific axioms. The second technical result of this paper is to present two equational theories for reasoning with digital circuits. The aim of the first is to be *sound and complete*: that is to say, equality in the equational theory is equivalent to equality of stream functions To do this, 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 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. We also present a 'global equation' for checking the equality of two circuits by checking a finite number of smaller circuits. This gives, for the first time, a definitive compositional theory of digital circuits.

This equational theory can be used to derive another, simpler, theory constructed primarily from 'local' equations which show how the components in a circuit interact with each other. This setting is more appropriate for an *operational semantics* for digital circuits: in particular, we show that the unpleasant situation of 'unproductive' circuits, i.e. circuits that cannot be evaluated via rewriting, can be avoided. Consequently, any circuit can be syntactically reduced to a (potentially infinite but eventually periodic) sequence of values.

Rather than restricting to the traditional one dimensional term syntax for monoidal categories, we use the two dimensional graphical syntax of *string diagrams* [JS91]. 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+22b; Bon+22c; Kay21]. While this connection is not examined closely in this paper, it means that the equational theory of digital circuits can be 'slotted in' to existing graph rewriting work with ease.

2 Digital circuits

Let us 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, $\bullet \in \mathcal{V}$ *is a* disconnected *value*, $\circ \in \mathcal{V}$ *is a* short-circuit *value*, $\mathcal{G}$ *is a (usually finite) set of* gate symbols, *and* $\# \colon \mathcal{G} \to \mathbb{N}$ *is an* arity *function.*

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.

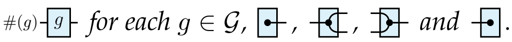
Example 2 (Gate-level circuits)**.** The signature for *gate-level circuits* is $\Sigma_B = (\mathcal{V}_B, \mathsf{n}, \mathsf{b}, \mathcal{G}_B, \#_B)$, where $\mathcal{V}_B := \{\mathsf{n}, \mathsf{f}, \mathsf{t}, \mathsf{b}\}$, respectively representing *no* signal, a *false* signal, a *true* signal and *both* signals at once, $\mathcal{G}_B := \{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\}$, and $\#_B := \mathrm{AND} \mapsto 2, \mathrm{OR} \mapsto 2, \mathrm{NOT} \mapsto 1$.

2.1 Syntax

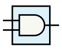
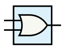
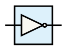
A circuit signature freely generates a monoidal category of *combinational circuits* and a symmetric traced monoidal category of *sequential circuits*. We employ the two dimensional syntax of *string diagrams* [JS91; JSV96; Sel11] as it is both intuitive and technically advantageous. Diagrams are written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

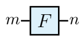
String diagrams are an especially natural language for *PROPs* [Lac04], symmetric monoidal categories with natural numbers as objects and addition as tensor product. A generator $m \to n$ is then drawn as a box with m input wires and n output wires. To avoid cluttering diagrams, wires may be collapsed into one wire and labelled appropriately. In this paper, drawing the wires in this way is purely notational. However, this idea has been formalised syntactically elsewhere [WGZ22].

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*

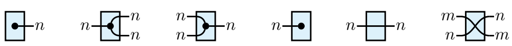


Rectangular light blue generators are *gates* for each gate symbol in the signature Σ. The remaining generators are *structural* generators for manipulating wires: these are present regardless of the signature. In order, they are for *introducing* wires, *forking* wires, *joining* wires and *stubbing* wires.

Example 4. The gate generators of $\mathbf{CCirc}_{\Sigma_B}$ are , , and .

When drawing circuits, the coloured backgrounds of generators will often be omitted in the interests of clarity. Since the category is freely generated, morphisms are defined by juxtaposing the generators in a given signature sequentially or in parallel with each other, the symmetry  and the identity . Arbitrary combinational circuit morphisms defined in this way are drawn as light boxes .

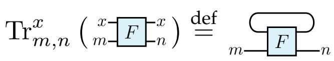
Notation 5. *It is a simple exercise to define the structural generators, the identity and symmetry on arbitrary bit inputs and outputs using the axioms of symmetric monoidal categories. In diagrams, these are drawn as their single-bit counterparts:*



Combinational circuits compute functions of their inputs, but have no internal state. Real-world circuits often involve *delay* and *feedback*: these are known as *sequential circuits*. To model feedback, extra structure must be added to the category of combinational circuits in the form of a *trace*.

Definition 6 ([JSV96; Has09])**.** *A* symmetric traced monoidal category, *often abbreviated as STMC, is a symmetric monoidal category* $\mathcal{C}$ *equipped with a family of functions* $\mathrm{Tr}^X_{A,B}(-) \colon \mathcal{C}(X \otimes A, X \otimes B) \to \mathcal{C}(X, Y)$ *satisfying the axioms of STMCs listed in Fig. 1.*

In string diagrams, the trace is represented by joining some of the inputs of a circuit to its outputs.



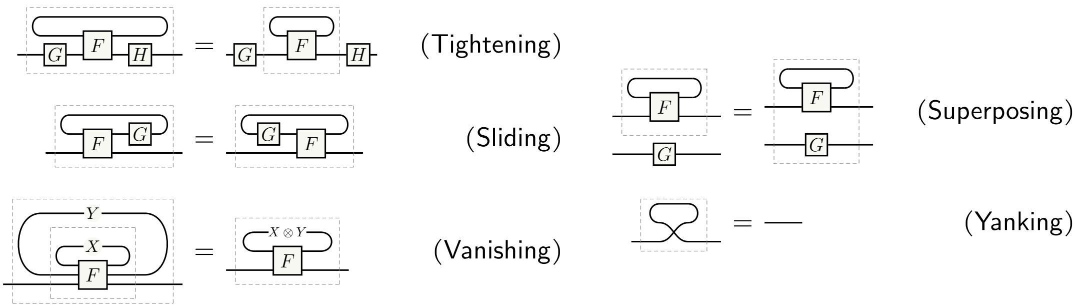
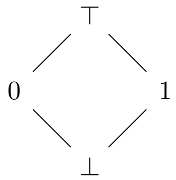


Figure 1: Equations that hold in any STMC.



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

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

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

Figure 2: The lattice structure on $\mathbf{V}_B$, and the truth tables of Belnap logic gates [Bel77].

Definition 7 (Sequential circuits)**.** *Let $\mathbf{SCirc}_\Sigma$ be the STMC freely generated over the generators of $\mathbf{CCirc}_\Sigma$ along with new generators $\boxed{v}-$ for each $v \in \mathcal{V} \setminus \bullet$, and $-\boxed{\Diamond}-$.*

Morphisms in $\mathbf{SCirc}_\Sigma$ are distinguished from those in $\mathbf{CCirc}_\Sigma$ by a darker green colouring. The additional generators introduce *state* into circuits. The smaller generators with no inputs are *instantaneous values*: these specify the initial state of a circuit. The diamond is a *delay* generator, which can be thought of as delaying its input by one unit of time.

Example 8. The value generators of $\mathbf{SCirc}_{\Sigma_B}$ are $\boxed{\mathsf{n}}-$, $\boxed{\mathsf{f}}-$, $\boxed{\mathsf{t}}-$ and $\boxed{\mathsf{b}}-$.

As with combinational circuits, sequential circuits are defined by juxtaposing generators. Arbitrary sequential circuit morphisms are drawn as dark green square boxes: ${}^{m}\!-\boxed{F}-^{n}$.

Notation 9. *As with the structural generators, it is useful to reason with multiple sequential generators in parallel: these will be drawn as $\boxed{\overline{v}}-^{n}$ and ${}^{n}\!-\boxed{\Diamond}-^{n}$, where $\overline{v} \in \mathcal{V}^n$.*

2.2 Semantics

We now interpret circuits in a semantic domain. Recall that a function f between two posets is *monotone* if $x \leq y \Rightarrow f(x) \leq f(y)$: monotone functions are the interpretation of combinational circuits. We write $\bot^n$ for the n-tuple containing only $\bot$ values.

Definition 10 (Interpretation)**.** *An interpretation of $\Sigma = (\mathcal{V}, \bullet, \circ, \mathcal{G}, \#)$ is a tuple $\mathcal{I} = (\mathbf{V}, [\![-]\!]^{\mathbf{V}}, [\![-]\!]^{\mathbf{G}})$ where $(\mathbf{V}, \sqsubseteq, \bot, \top)$ is a finite lattice, $[\![-]\!]^{\mathbf{V}}$ is a function $\mathcal{V} \to \mathbf{V}$, and $[\![-]\!]^{\mathbf{G}}$ is a map sending each $g \in \mathcal{G}$ to a monotone function $\mathbf{V}^{\#(g)} \to \mathbf{V}$. These functions are required to satisfy $[\![\bullet]\!]^{\mathbf{V}} = \bot$; $[\![\circ]\!]^{\mathbf{V}} = \top$; $[\![g]\!]^{\mathbf{G}}(\bot^m) = \bot$; and $[\![g]\!]^{\mathbf{G}}(\overline{v})$ is in the image of $[\![-]\!]^{\mathbf{V}}$ for all $\overline{v} \in \mathbf{V}^m$.*

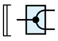
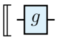
Example 11. Recall the signature $\Sigma_B = (\mathcal{V}_B, \mathsf{n}, \mathsf{b}, \mathcal{G}_B, \#_B)$ from Example 2. The values are interpreted in the four value lattice $\mathbf{V}_B = \{\bot, 0, 1, \top\}$, where $0 \sqcup 1 = \top$ and $0 \sqcap 1 = \bot$. The gates are interpreted using Belnap logic gates [Bel77]: the truth tables are listed in Fig. 2. Let $[\![-]\!]^{\mathbf{V}}_B := \{\mathsf{n} \mapsto \bot, \mathsf{f} \mapsto 0, \mathsf{t} \mapsto 1, \mathsf{b} \mapsto \top\}$ and $[\![-]\!]^{\mathbf{G}}_B := \{\mathrm{AND} \mapsto \wedge, \mathrm{OR} \mapsto \vee, \mathrm{NOT} \mapsto \neg\}$. Then the Belnap interpretation is defined as $\mathcal{I}_B = (\mathbf{V}_B, [\![-]\!]^{\mathbf{V}}_B, [\![-]\!]^{\mathbf{G}}_B)$. The astute reader may observe that $\wedge$ and $\vee$ are in fact the meet and join of another lattice structure on $\{\bot, 0, 1, \top\}$, in which 1 is the supremum, 0 is the infimum, and $\bot$,$\top$ occupy the wings.

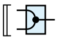
The lattice does not have to be as simple as $\mathbf{V}_B$. For example, it could contain 'weak' and 'strong' versions of the values, modelling those used in metal-oxide-semiconductor field-effect transistors (MOSFET). For the remainder of this paper, we fix an arbitrary interpretation $\mathcal{I} = (\mathbf{V}, [\![-]\!]^{\mathbf{V}}, [\![-]\!]^{\mathbf{G}})$.

Semantics are expressed formally using a *PROP morphism*, a strict symmetric monoidal functor between PROPs that is the identity on objects. When the domain of a functor is freely generated, it can be defined solely by its action on the generators.

Definition 12. *Let $\mathbf{Func}_{\mathcal{I}}$ be the (non-traced) PROP with morphisms $m \to n$ the monotone functions $\mathbf{V}^m \to \mathbf{V}^n$.*

Definition 13. *Let $[\![-]\!]^{\mathbf{C}}_{\mathcal{I}} : \mathbf{CCirc}_{\Sigma} \to \mathbf{Func}_{\mathcal{I}}$ be the PROP morphism with its action defined as*

$[\![$  $]\!]^{\mathbf{C}}_{\mathcal{I}} := (v) \mapsto ()$ $\quad$ $[\![$  $]\!]^{\mathbf{C}}_{\mathcal{I}} := (v) \mapsto (v, v)$ $\quad$ $[\![$  $]\!]^{\mathbf{C}}_{\mathcal{I}} := \overline{x} \mapsto [\![g]\!]^{\mathbf{G}}(\overline{x})$

$[\![$  $]\!]^{\mathbf{C}}_{\mathcal{I}} := () \mapsto (\bot)$ $\quad$ $[\![$  $]\!]^{\mathbf{C}}_{\mathcal{I}} := (v, w) \mapsto (v \sqcup w)$

Sequential circuits are more complicated, as their output may depend on previous inputs. Their inputs are thus *streams*, infinite sequences of values. Given a set X, we denote the set of streams of X by X^ω. A stream can also be viewed as a function $\mathbb{N} \to X$; consequently we write $\sigma(k)$ for the kth element of a stream $\sigma \in X^\omega$. There are two important operations used to reason with streams.

Definition 14 (Operations on streams)**.** *The* initial value *is a function $\mathsf{i}(-) \colon X^\omega \to X := \sigma \mapsto \sigma(0)$, producing the 'head' of a stream; and the* stream derivative *is a a function $\mathsf{d}(-) \colon X^\omega \to X^\omega := \sigma \mapsto (i \mapsto \sigma(i+1))$, producing its 'tail'.*

These operations can define streams: for an element $x \in X$ and stream $\sigma \in X^\omega$, the stream $x :: \sigma$ is the unique stream with initial value x and stream derivative σ.

Sequential circuits are semantically interpreted as *stream functions*: these consume streams of inputs and produce streams of outputs. Given a stream function f and some input stream σ, the kth element of its output stream is written $f(\sigma)(k)$.

In particular, we will define the semantics of a circuit as a causal (Definition 15), monotone (Definition 18) stream function with finitely many stream derivatives (Definition 17).

The output of a circuit depends on the inputs it has seen 'so far'. We call this notion *causality*.

Definition 15 (Causal stream function [Rut06])**.** *A stream function $f \colon M^\omega \to N^\omega$ is* causal *if for all $i \in \mathbb{N}$ and all $\sigma, \tau \in M^\omega$ with the property that $\sigma(j) = \tau(j)$ for all $j \leq i$ it holds that $f(\sigma)(i) = f(\tau)(i)$.*

A stream function being causal means the ith element of the output stream depends only on inputs 0 through i. An important consequence of this is that the initial value and derivative operations defined for streams can be extended to *causal* stream functions.

Definition 16 (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 arbitrary $\sigma \in M^\omega$. The* functional stream derivative of f on input a *is the function $\partial_a f \colon M^\omega \to N^\omega := \sigma \mapsto \mathsf{d}(f(a :: \sigma))$. We may abbreviate $\partial_a f$ to f_a in the sequel.*

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

Definition 17. *For a causal stream function $f \colon M^\omega \to N^\omega$, we define f_w for $w \in M^\star$ by induction on the length of w. If w is the empty word, $f_w = f$. Otherwise we can write $w = aw'$, in which case $f_w = \partial_a f_{w'}$.*

We say f has finitely many stream derivatives *if the set $\{f_w : w \in M^\star\}$ is finite.*

Since circuits are built from components whose interpretations are monotone functions, their interpretations as stream functions must also be monotone.

Definition 18. *Let M be a partially ordered set. If $\sigma, \tau \in M^\omega$, we say $\sigma \leq_{M^\omega} \tau$ if $\sigma(k) \leq_M \tau(k)$ for all $k \in \mathbb{N}$. This relation defines a partial ordering on M^ω.*

Next, let N be another partially ordered set and $f, g \colon M^\omega \to N^\omega$ be functions. We say $f \preceq g$ if $f(\sigma) \leq_{N^\omega} g(\sigma)$ for all $\sigma \in M^\omega$. This defines a partial ordering on the set of functions from M^ω to N^ω.

Finally, a causal stream function $f \colon M^\omega \to N^\omega$ is monotone *if it is monotone with respect to the above orderings on M^ω and N^ω.*

We may drop the subscripts on these orderings when they are obvious from context. It is now possible to assemble a category of stream functions that correspond to sequential circuits. We will construct an STMC of causal monotone stream functions as the semantic domain.

For the monoidal structure, 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 $\sigma \in (X^{m+n})^\omega$, we write $\pi_0(\sigma) \in (X^m)^\omega$ for the stream of tuples containing the first m elements of tuples in σ, and $\pi_1(\sigma) \in (X^n)^\omega$ for the stream of tuples containing the last n elements.

Lemma 19. *Causality, monotonicity and having finitely many stream derivatives is 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 finitely many stream derivatives, both the composition and product of two stream functions f and g, the largest the set of stream derivatives could be is the product of stream derivatives of f and g, so this will also be finite. Finally, the composition and product of any monotone function is monotone. □

Definition 20. *Let* $\mathbf{Stream}_\mathcal{I}$ *be the PROP with morphisms* $m \to n$ *the causal, monotone stream functions* $f\colon (\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$ *with finitely many stream derivatives.*

Perhaps surprisingly, $\mathbf{Stream}_\mathcal{I}$ is a traced category.

Proposition 21. *Let* $f\colon (\mathbf{V}^{x+m})^\omega \to (\mathbf{V}^{x+n})^\omega$ *be a morphism in* $\mathbf{Stream}_\mathcal{I}$*. For each* $\sigma \in (\mathbf{V}^m)^\omega$*, there is a monotone endofunction on* $(\mathbf{V}^x)^\omega$ *given by* $\tau \mapsto \pi_0(f(\tau +\!\!+\, \sigma))$*; let* $\mu_f(\sigma)$ *be the least fixed point of this function. The mapping* $\sigma \mapsto \mu_f(\sigma)$ *is itself a morphism of* $\mathbf{Stream}_\mathcal{I}$*. Then a trace* $\mathrm{Tr}^x(f)\colon (\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$ *is defined by* $(\mathrm{Tr}^x(f))(\sigma) := \pi_1(f(\mu_f(\sigma) +\!\!+\, \sigma))$*.*

Proof. Since f is a morphism of $\mathbf{Stream}_\mathcal{I}$, it has finitely many stream derivatives. For each of these stream derivatives, we define $\widehat{f_w}\colon (\mathbf{V}^{x+m})^\omega \to (\mathbf{V}^x)^\omega$ to be $\tau +\!\!+\, \sigma \mapsto \pi_0(\partial_w f(\tau +\!\!+\, \sigma))$. Note that each of these functions are causal and monotone.

By the Kleene fixed point theorem, the least fixed point of $\widehat{f}(\sigma, -)$ can be obtained by composing $\widehat{f}(\sigma, -)$ repeatedly with itself. In particular, $\mu_f(\sigma) = \bigvee_{k \in \omega} \widehat{f^k}(\sigma, \bot^\omega)$ where $\widehat{f^k}(\sigma, \tau) = \widehat{f}(\sigma, \widehat{f^{k-1}}(\sigma, \tau))$ is the k-fold composition of $\widehat{f}(\sigma, -)$ with itself with $\widehat{f^0}(\sigma, \tau) = \tau$. That the mapping $\sigma \mapsto \mu_f(\sigma)$ is causal and monotone is straightforward: each of the functions in the join is causal and monotone, and join preserves these properties. It remains to show this mapping has finitely many stream derivatives.

The set $\{\widehat{f_w} : w \in (\mathbf{V}^{x+m})^*\}$ is a finite subset of the poset of functions $(\mathbf{V}^{x+m})^\omega \to (\mathbf{V}^x)^\omega$ under $\preceq$ (Definition 18). Restricting the ordering $\preceq$ to this set yields a finite poset. Since this poset is finite, the set of strictly increasing sequences in this poset is also finite. We exhibit a surjection from the set of strictly increasing sequences in this poset to stream derivatives of μ_f.

Suppose $S = \widehat{f_{w_1}} \prec \widehat{f_{w_2}} \prec \cdots \prec \widehat{f_{w_\ell}}$ is a strictly increasing sequence in $\{\widehat{f_w} : w \in (\mathbf{V}^{x+m})^*\}$. We define a function $(\mathbf{V}^x)^\omega \to (\mathbf{V}^x)^\omega$ by $g_S(\sigma) = \bigvee_{k \in \omega} g_k(\sigma, \bot^\omega)$ where

$$g_k(\sigma, \tau) = \begin{cases} \tau & \text{if } k = 0 \\ \widehat{f_{w_k}}(\sigma, g_{k-1}(\sigma, \tau)) & \text{if } 1 \le k \le \ell \\ \widehat{f_{w_\ell}}(\sigma, g_{k-1}(\sigma, \tau)) & \text{if } \ell < k \end{cases}.$$

Note μ_f is g_S where S is the one-item sequence $\widehat{f}$. We show the set of functions

$$\{g_S : S \text{ a strictly increasing sequence}\}$$

is closed under stream derivative, and so μ_f has finitely many stream derivatives.

$$\begin{aligned}
(\partial_{a +\!\!+ b} \widehat{f_w})(\sigma, \tau) &= \mathsf{d}(\widehat{f_w}(a :: \sigma, b :: \tau)) \\
&= \mathsf{d}(\pi_0(\partial_w f((b :: \tau) +\!\!+ (a :: \sigma)))) \\
&= \pi_0(\mathsf{d}(\partial_w f((b +\!\!+ a) :: (\tau +\!\!+ \sigma)))) \\
&= \pi_0(\partial_{b +\!\!+ a} \partial_w f(\tau +\!\!+ \sigma)) \\
&= \pi_0(\partial_{(b +\!\!+ a) w} f(\tau +\!\!+ \sigma))
\end{aligned}$$

□

The semantics of circuits in $\mathbf{SCirc}_\Sigma$ can now be defined as stream functions: we first define the necessary *stateful* functions.

Definition 22. *For each $v \in \mathbf{V}$, let $\mathsf{val}_v \colon (\mathbf{V}^0)^\omega \to \mathbf{V}^\omega$ be defined as $\mathsf{val}_v()(0) := v$ and $\mathsf{val}_v()(k+1) := \bot$. Let $\mathsf{shift} \colon \mathbf{V}^\omega \to \mathbf{V}^\omega$ be defined as $\mathsf{shift}(\sigma)(0) := \bot$ and $\mathsf{shift}(\sigma)(k+1) := \sigma(k)$.*

Lemma 23. *The stream functions in Definition 22 are causal, monotone and have finitely many stream derivatives.*

Proof. For the first four functions, the kth element of the output stream is computed by a monotone operation on kth element of the input stream, so the stream function is monotone and causal. Since the kth input element cannot affect a later output element, there is one stream derivative: the original function. The function val_v has no inputs so it is trivially causal and monotone. It has one stream derivative: the stream function that constantly outputs $\bot$. The function shift is causal as the $i+1$th output element depends on only the ith element. There are $|\mathbf{V}|$ stream derivatives, as there is a different one for each possible input value. Each of these stream derivatives is monotone, as the initial output is fixed regardless of input, and, on input a, the stream derivative is the stream function that initially outputs a. □

These stream functions are therefore morphisms in $\mathbf{Stream}_\mathcal{I}$, so are suitable candidates for the semantics of generators in $\mathbf{SCirc}_\Sigma$.

Definition 24. *Let $[\![-]\!]^{\mathbf{S}}_{\mathcal{I}} : \mathbf{SCirc}_\Sigma \to \mathbf{Stream}_\mathcal{I}$ be the traced PROP morphism with its action defined as*

$$[\![-\boxed{F}-]\!]^{\mathbf{S}}_{\mathcal{I}}(\sigma)(k) := [\![-\boxed{F}-]\!]^{\mathbf{C}}_{\mathcal{I}}(\sigma(k)) \qquad [\![\boxed{v}-]\!]^{\mathbf{S}}_{\mathcal{I}} := \mathsf{val}_v \qquad [\![-\boxed{\diamond}-]\!]^{\mathbf{S}}_{\mathcal{I}} := \mathsf{shift}$$

Given a sequential circuit $-\boxed{F}-$, we say that the stream function $[\![-\boxed{F}-]\!]^{\mathbf{S}}_{\mathcal{I}}$ is its behaviour under $\mathcal{I}$.

3 Mealy machines

Every circuit in $\mathbf{SCirc}_\Sigma$ has a corresponding stream function in $\mathbf{Stream}_\mathcal{I}$, computed using $[\![-]\!]^{\mathbf{S}}_{\mathcal{I}}$. We now turn our attention to the converse: does every stream function $f \in \mathbf{Stream}_\mathcal{I}$ have a (set of) circuits in $\mathbf{SCirc}_\Sigma$ such that f is their behaviour?

To answer this, we will view digital circuits as *Mealy machines* [Mea55], which are often used to specify the behaviour of digital circuits [KJ09]. Mealy machines are also interesting because of their *coalgebraic* properties: every Mealy machine has a unique corresponding causal stream function.

3.1 Mealy machines and coalgebra

Definition 25 (Mealy machine [Mea55])**.** *Let M and N be finite sets. A (finite)* Mealy machine *(with interface (M, N)) is a tuple (S, f, s_0) where S is a finite set called the* state space*, $f \colon S \to (N \times S)^M$ is the* Mealy function*, and $s_0 \in S$ is the* start state*.*

The sets M and N are the *input* and *output* spaces of the machine. Given a state $s \in S$ and input $a \in M$, the Mealy function f produces a pair $f(s)(a) = \langle n, s' \rangle$. We will also use the shorthand $f_O := (s, a) \mapsto \pi_0(f(s)(a))$ and $f_T := (s, a) \mapsto \pi_1(f(s)(a))$ for the output and state transition component of the Mealy function respectively.

Mealy machines can be viewed *coalgebraically*. Recall that a coalgebra of an endofunctor F is a pair of an object X and a morphism $X \to FX$. Then, the first two components of any Mealy machine (S, f) with interface (M, N) is a coalgebra of the functor $Y \colon \mathbf{Set} \to \mathbf{Set}$, defined as $S \mapsto (N \times S)^M$. We call this a *Mealy coalgebra*; it is essentially the same as a Mealy machine but without a designated start state.

Example 26. In [BRS08], the notation $f(s)(a) = \langle s[a], s_a \rangle$ is also used to describe the Mealy function, which coincides with the notation used for stream functions: this is not a coincidence! Let Γ be the set of causal stream functions $M^\omega \to N^\omega$ for sets M and N, and let $\nu \colon \Gamma \to (N \times \Gamma)^M$ be the function defined as $\nu : (f, a) \mapsto \langle f[a], f_a \rangle$. Then (Γ, ν, f) is a Mealy coalgebra with interface (M, N). In fact, Proposition 28 shows it is the *final* Mealy coalgebra.

A homomorphism h between two Mealy coalgebras (S, f) and (T, g) with interface (M, N) is a function $h \colon S \to T$ preserving outputs and transitions, i.e. for a Mealy function f, $h\,(f_O(s, a)) = f_O(s, a)$ and $h\,(f_T(s, a)) = f_T(h(s), a)$.

Definition 27. *For a Mealy machine (S, f) and $i \in \mathbb{N}$, let $T_{f,i} \colon S \times M^\omega \to S$ be defined as $T_{f,0}(s, \sigma) := s$ and $T_{f,k+1}(s, \sigma) := f_T(T_{f,i}(s, \sigma), \sigma(i))$.*

Proposition 28 (Proposition 2.2, [Rut06])**.** *For every Mealy coalgebra (S, f) with interface (M, N), there exists a unique homomorphism $!(-)\colon (S, f) \to (\Gamma, \nu)$.*

Proof. Given initial state $s \in S$ and stream $\sigma \in (\mathbf{V}^m)^\omega$, $!(S, f)(s)(\sigma)(i) := f_O(T_{f,i}(s, \sigma), \sigma(i))$. □

This means each Mealy machine (S, f, s_0) has a corresponding stream function $!(S, f)(s_0)$. Given some input stream σ, the elements of the output stream $!(S, f)(s_0)(\sigma)$ are the outputs that (S, f, s_0) would produce given the same inputs.

3.2 Circuits as Mealy machines

Not all Mealy machines defined in this way correspond to a digital circuit in $\mathbf{SCirc}_\Sigma$, as the lattice structure on the values must be taken into account. The Mealy machines that *do* have a corresponding circuit can be identified by checking whether $!(-)$ lands in $\mathbf{Stream}_\mathcal{I}$. By definition, all stream functions in the image of $!(-)$ are causal. By virtue of reasoning with *finite* Mealy machines we can also conclude the following:

Lemma 29. *Any stream function in the image of $!(-)$ has finitely many stream derivatives.*

Proof. S is finite, and $!(-)$ must preserve transitions. □

Monotonicity is more subtle, as the states of a Mealy machine are not naturally ordered. However, since each state corresponds to a stream function, they can inherit the ordering from Definition 18, and subsequently a notion of monotonicity.

Definition 30 (State order)**.** *Given a Mealy machine (S, f, s_0) with interface (M, N) where M and N are partial orders, we say that, for states $s, s' \in S$, $s \preceq s'$ if $!(S, f)(s) \preceq\, !(S, f)(s')$.*

Definition 31 (Monotone Mealy machine)**.** *A Mealy machine (S, g, s_0) with interface (M, N) where $(M, \leq_M)$ and $(N, \leq_N)$ are partial orders, is called* monotone *if g_O is monotone with respect to $\leq_M$ and $\leq_N$, and g_T is monotone with respect to $\leq_M$ and $\preceq$.*

Lemma 32. *Given a causal stream function $g\colon M^\omega \to N^\omega$, the functions $a \mapsto g[a]$ and $a \mapsto g_a$ are monotone if and only if g is monotone.*

Proof. First the $(\Leftarrow)$ direction. Recall that monotonicity implies that, for a given stream function g and inputs σ, τ, if $\sigma \leq \tau$ i.e. if $\sigma(i) \leq \tau(i)$ for all $i \in \mathbb{N}$, then $g(\sigma) \leq g(\tau)$ i.e. $g(\sigma)(i) \leq g(\tau)(i)$. Observe that this means that $g(a :: \sigma)(i) \leq g(b :: \sigma)(i)$ if $a \leq b$. Using this fact it is a simple exercise to show that the $a \mapsto g[a]$ and $a \mapsto g_a$ are monotone. Let $a, b \in M$ such that $a \leq b$. First we show $a \mapsto g[a]$ is monotone:

$$\begin{aligned} g[a] &= \mathsf{i}(g(a :: \sigma)) \\ &= g(a :: \sigma)(0) \\ &\leq g(b :: \sigma)(0) && \text{by monotonicity of } g \\ &= \mathsf{i}(g(b :: \sigma)) \\ &= g[b] \end{aligned}$$

And now we show that $a \mapsto g_a$ is monotone:

$$\begin{aligned} g_a(\sigma)(i) &= \mathsf{d}(g(a :: \sigma))(i) \\ &= g(a :: \sigma)(i+1) \\ &\leq g(b :: \sigma)(i+1) && \text{by monotonicty of } g \\ &= \mathsf{d}(g(b :: \sigma))(i) \\ &= g_b(\sigma)(i) \end{aligned}$$

Now the $(\Rightarrow)$ direction. For a stream function g, assume that $a \mapsto g[a]$ and $a \mapsto g_a$ are monotone. We need to show that g is monotone, i.e. for streams σ, τ, if $\sigma \leq \tau$ then $g(\sigma)(i) \leq g(\tau)(i)$. Let $\sigma := w :: a :: \sigma'$ and $\tau := w' :: b :: \tau'$ where w is a finite sequence of length k. Since $\sigma \leq \tau$, $w \leq w'$ and $a \leq b$. Let g_w be the result of repeatedly obtaining the stream derivative for each element of w: this is also monotone by function composition. Therefore:

$$
\begin{aligned}
g(\sigma) &= g(w :: a :: \sigma')(k) \\
&= \mathsf{d}^k(g(w :: a :: \sigma'))(0) \\
&= g_w(a :: \sigma')(0) \\
&= g_w[a] \\
&\leq g_w[b] && \text{by monotonicity of } a \mapsto g[a] \\
&= g_w(b :: \tau)(0) \\
&\leq g_{w'}(b :: \tau)(0) && \text{by monotonicity of } a \mapsto g_a \\
&= \mathsf{d}^k(g(w' :: b :: \sigma'))(0) \\
&= g(w' :: \tau')(k) \\
&= g(\tau)
\end{aligned}
$$

□

Proposition 33. *For a monotone Mealy machine (S, g, s_0), $!(S, f)(s_0) \in \mathbf{Stream}_{\mathcal{I}}$.*

Proof. $!(S, f)(s)$ is causal by definition, and has finitely many stream derivatives by Lemma 29. Since the initial output and stream derivative of $!(S, f)(s)$ is defined as $a \mapsto s[a]$ and $a \mapsto s_a$ respectively: as these are monotone by definition of a monotone Mealy machine, the stream function is monotone by Lemma 32. □

To map from sequential circuits into monotone Mealy machines, the latter must also be assembled into a traced prop. There are standard notions of composition for Mealy machines:

Definition 34 (Composition of Mealy machines [Gin14])**.** *For Mealy machines (S, f, s_0) with interface (M, N) and (T, g, t_0) with interface (N, P), their* cascade product *is a Mealy machine over (M, P), with state set $S \times T$, Mealy function*

$$((s, t), a) \mapsto \langle g_O(t, f_T(s, a)), (f_T(s, a), g_T(t, f_T(s, a))) \rangle$$

and initial state (s', t'). For Mealy machines (S, f, s_0) with interface (M, N) and $h(T, g, t_0)$ with interface (P, Q), their direct product *is a Mealy machine over $(M \times P, N \times Q)$ with state set $S \times T$, Mealy function*

$$((s, t), (a, b)) \mapsto \langle (f_O(s, a), g_O(t, b)), (f_T(s, a), g_T(t, b)) \rangle$$

and initial state (s_0, t_0).

The desired PROP is constructed by setting the interfaces of Mealy machines to powers of $\mathbf{V}$.

Definition 35. *Let $\mathbf{Mealy}_{\mathcal{I}}$ be the PROP with morphisms $m \to n$ as the monotone Mealy machines with interface $(\mathbf{V}^m, \mathbf{V}^n)$. Composition is by cascade product and tensor on morphisms is by direct product.*

Using the same reasoning as with Proposition 21, a trace operator can be defined on monotone Mealy machines.

Definition 36 (Least fixed point of Mealy machines)**.** *Let (S, g, s_0) be a monotone Mealy machine over $(\mathbf{V}^{x+m}, \mathbf{V}^{x+n})$ and let μ_a be the fixpoint of the function $r \mapsto g(r, a)$ for input $a \in \mathbf{V}^m$. Then the* least fixed point *of A is a Mealy machine over $(\mathbf{V}^m, \mathbf{V}^n)$ defined as $(S, (s, a) \mapsto g(\mu_a, a), s_0)$.*

Proposition 37. *The least fixed point operation is a trace operator on $\mathbf{Mealy}_{\mathcal{I}}$.*

Proof. Since Mealy machines in $\mathbf{Mealy}_{\mathcal{I}}$ are monotone, their Mealy functions are also monotone, so they have a least fixed point. The axioms of STMCs can be shown to hold with this construction. □

Definition 38. *Let $[-]_{\mathcal{I}} : \mathbf{SCirc}_{\Sigma} \to \mathbf{Mealy}_{\mathcal{I}}$ be the traced PROP morphism with action*

$$[\boxed{F}]_{\mathcal{I}} := (\emptyset, \overline{v} \mapsto \langle [\![\boxed{F}]\!]^{\mathbf{C}}_{\mathcal{I}}(\overline{v}), () \rangle, ())$$

$$[\boxed{v}]_{\mathcal{I}} := (\{s_v, s_\perp\}, \{s_v \mapsto \langle v, s_1 \rangle, s_\perp \mapsto \langle \perp, s_1 \rangle\}, s_v)$$

$$[\boxed{\diamond}]_{\mathcal{I}} := (\{s_v \mid v \in \mathbf{V}\}, (s_v, a) \mapsto \langle s_a, v \rangle, s_\perp)$$

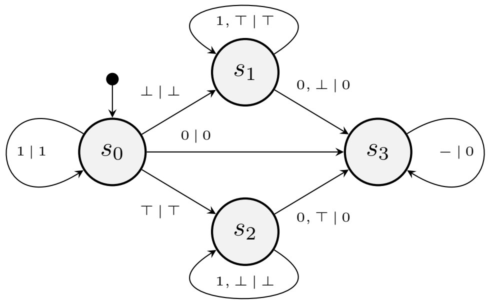
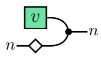
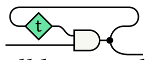
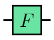
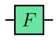


Figure 3: The Mealy machine constructed in Example 40, where $v \mid w$ means the output w is produced on input v.

Before proceeding, we will introduce some notation for a register with an initial value, which will be a common construction used in the remainder of the paper.

Notation 39. *For $\overline{v} \in \mathbf{V}^n$, let*  $:=$ .

Example 40. Consider the circuit . This circuit has one delay and one value, so the Mealy machine obtained by using $[-]_{\mathcal{I}}$ will have eight states. However, some of these states are inaccessible: the accessible states are shown in Fig. 3 with their outputs and transitions, The initial state is s_0.

For Mealy machines to be useful in our context, it is essential that translation between circuits and Mealy machines *preserves behaviour*. That is to say, if we translate a circuit  into a Mealy machine and then into a stream function, this stream function should be the behaviour of .

To show this, we must prove some lemmas about the composition of Mealy machines.

Lemma 41. *For Mealy machines (S, f, s_0) and (T, g, t_0), let $(S, f, s_0) \mathbin{⨾} (T, g, t_0) := (S \times T, h, (s_0, t_0)$ be their composition. Then $T_{h,i}((s,t), \sigma) = (T_{f,i}(s, \sigma), T_{g,i}(t, !(S, f)(s)(\sigma)))$.*

Proof. This is by induction over the index i. When $i = 0$, the statement holds immediately. Now we show $i = k + 1$.

$$\begin{aligned}
T_{h,k+1}((s,t),\sigma) &= h_1(T_{h,k}((s,t),\sigma), \sigma(k)) \\
&= h_1((T_{f,k}(s,\sigma), T_{g,k}(t, !(S,f)(s)(\sigma))), \sigma(i)) \\
&:= (f_1(T_{f,k}(s,\sigma), \sigma(k)), g_1(T_{g,k}(t, !(S,f)(s)(\sigma)), f_0(T_{f,k}(s,\sigma), \sigma(k)))) \\
&:= (T_{f,k+1}(s,\sigma), g_1(T_{g,k}(t, !(S,f)(s)(\sigma)), f_0(T_{f,k}(s,\sigma), \sigma(k)))) \\
&:= (T_{f,k+1}(s,\sigma), g_1(T_{g,k}(t, !(S,f)(s)(\sigma)), !(S,f)(s)(\sigma)(k))) \\
&:= (T_{f,k+1}(s,\sigma), T_{g,k+1}(t, !(S,f)(s)(\sigma)))
\end{aligned}$$

□

Proposition 42. *For Mealy machines (S, f, s_0) and $B = (T, g, t_0)$, $!(A \mathbin{⨾} B)(s,t) = !(T,g)(t) \circ !(S,f)(s)$.*

Proof. Given $i \in \mathbb{N}$, $!(S,f)(s)(\sigma)(i) := f_O(T_{f,i}(s,\sigma), \sigma(i))$ and $!(T,g)(t)(\tau)(i) := g_O(T_{g,i}(t,\tau), \tau(i))$. Recall that the Mealy function of $A \mathbin{⨾} B$ is $h := ((s,t),a) \mapsto \langle g_O(t, f_O(s,a)), (f_T(s,a), g_T(t, f_O(s,a))) \rangle$. This means that

$$\begin{aligned}
!A \mathbin{⨾} B(s,t)(\sigma)(i) &:= h_O(T_{h,i}((s,t),\sigma), \sigma(i)) \\
&= (((s,t),a) \mapsto g_O(t, f_O(s,a)))(T_{h,i}((s,t),\sigma), \sigma(i)) \\
&= (((s,t),a) \mapsto g_O(t, f_O(s,a)))((T_{f,i}(s,\sigma), T_{g,i}(t, !(S,f)(s)(\sigma))), \sigma(i)) \\
&= g_O(T_{g,i}(t, !(S,f)(s)(\sigma)), f_O(T_{f,i}(s,\sigma), \sigma(i)))
\end{aligned}$$

Now we consider the other side of the equation. Since composition in $\mathbf{Stream}_{\mathcal{I}}$ is function composition, this means that

$$\begin{aligned}
(!(T,g)(t) \circ !(S,f)(s))(i) &= (\tau \mapsto !(T,g)(t)(\tau(i)))(!(S,f)(s)) \\
&:= (\tau \mapsto g_O(T_{g,i}(t,\tau), \tau(i)))(!(S,f)(s)) \\
&= g_O(T_{g,i}(t, !(S,f)(s)), !(S,f)(s)(i)) \\
&:= g_O(T_{g,i}(t, !(S,f)(s)), f_O(T_{f,i}(s,\sigma), \sigma(i)))
\end{aligned}$$

So we can conclude that $!A \fatsemi B(s,t)(\sigma) = (!(T,g)(t) \circ !(S,f)(s))$. $\square$

Theorem 43. $[\![-]\!]^{\mathbf{S}}_{\mathcal{I}} = !(-) \circ [-]_{\mathcal{I}}$.

Proof. This is by induction over the structure of $\boxed{F}$. The combinational generators are easy as their streams have one stream derivative and their Mealy machines have one state. The sequential components are more interesting. For values:

$$[\![\boxed{v}]\!]^{\mathbf{S}}_{\mathcal{I}}(0) = v \qquad [\![\boxed{v}]\!]^{\mathbf{S}}_{\mathcal{I}}(i+1) = \bot \qquad [\boxed{v}]_{\mathcal{I}} := (\{s_v, s_\bot\}, s_w \mapsto \langle w, s_\bot \rangle, s_v)$$

Let $f := s_w \mapsto \langle w, s_\bot \rangle$. Then, $!\,[\![\boxed{v}]\!]^{\mathbf{S}}_{\mathcal{I}}(0) := f_O(s_v) = v$, and $!\,[\![\boxed{v}]\!]^{\mathbf{S}}_{\mathcal{I}}(i+1) := f_O(T_{f,i}(s_v)) = f_O(s_\bot) = \bot$, so all the elements of the stream agree. For delay:

$$[\![\boxed{\diamond}]\!]^{\mathbf{S}}_{\mathcal{I}}(\sigma)(0) = \bot \qquad [\![\boxed{\diamond}]\!]^{\mathbf{S}}_{\mathcal{I}}(\sigma)(i+1) = \sigma(i) \qquad [\boxed{\diamond}]_{\mathcal{I}} := (\{s_v \mid v \in \mathbf{V}\}, (s_w, a) \mapsto \langle w, s_a \rangle, s_\bot)$$

Let $g := (s_w, a) \mapsto \langle w, s_a \rangle$. Then, $!\,[\![\boxed{\diamond}]\!]^{\mathbf{S}}_{\mathcal{I}}(\sigma)(0) = g_O(s_w) = \bot$, which agrees with $[\![\boxed{\diamond}]\!]^{\mathbf{S}}_{\mathcal{I}}(\sigma)(0)$. Now consider $!\,[\![\boxed{\diamond}]\!]^{\mathbf{S}}_{\mathcal{I}}(\sigma)(i+1) = g_O(T_{g,i}(s_\bot, \sigma), \sigma(i+1))$. By definition of g, $T_{g,i}(s_\bot, \sigma) = s_v$ when $\sigma(i)$ does. So $g_O(T_{g,i}(s_\bot, \sigma), \sigma(i+1)) = g_O(s_{\sigma(i)}, \sigma(i+1)) = \sigma(i)$. So the stream functions agree here too.

Since the statement holds for the generators, we just need to show that the action of the operations is preserved. Composition is preserved by Proposition 42. Tensor is trivial as it is direct product. The trace is computed as the least fixpoint in both settings. So $[\![-]\!]^{\mathbf{S}}_{\mathcal{I}} = !(-) \circ [-]_{\mathcal{I}}$.

$\square$

4 Circuit synthesis

We have shown that, in addition to interpreting circuits directly as causal monotone stream functions with finitely many stream derivatives, we can also interpret them as (monotone) Mealy machines. Exploiting the coalgebraic structure of Mealy machines, we will now define a map back from a stream function f into a circuit with f as its behaviour.

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 44 (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 $\langle f \rangle_{\mathcal{I}} = (S, g)$ where $g(h)(a) = \langle h[a], h_a \rangle$, has the smallest state space of Mealy machines with the property $!\,\langle f \rangle_{\mathcal{I}} = f$.*

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 45. Recall the circuit from Example 40. The corresponding stream function $f\colon \mathbf{V}^\omega \to \mathbf{V}^\omega$ is defined for input σ and $k \in \mathbb{N}$ as

$$f(\sigma)(0) = \sigma(0) \qquad f(\sigma)(k+1) = f(\sigma)(k) \wedge \sigma(k+1)$$

We shall now recover a (finite) Mealy machine from this function. Inputting each value in $\mathbf{V}$ yields four stream derivatives: $f_\bot, f_0, f_1$ and $f_\top$. However, f_1 is actually equal to f, as 1 is the unit for $\wedge$. So we have states $\{f, f_\bot, f_\top, f_0\}$. By taking further stream derivatives it can be shown that these states completely specify the corresponding Mealy machine: in fact, it is the Mealy machine drawn in Example 40 where $f := s_0 . f_\bot := s_1, f_\top := s_2$, and $f_0 := s_3$.

Before proceeding with retrieving a circuit, it is essential to check if the recovered Mealy machine is in fact a *monotone* Mealy machine, i.e. it is in $\mathbf{Mealy}_{\mathcal{I}}$. Recall that by Lemma 32, given a fixed stream function f we know that $a \mapsto f[a]$ and $a \mapsto f_a$ are monotone. Now we must check the same holds when we fix the input.

Lemma 46. *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 $\langle f \rangle_{\mathcal{I}} = (S, g)$ defined as in Corollary 44, the Mealy function g is monotone.*

Proof. Lemma 32 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. $\square$

Proposition 47. *$\langle - \rangle_{\mathcal{I}}$ is a PROP morphism $\mathbf{Stream}_{\mathcal{I}} \to \mathbf{Mealy}_{\mathcal{I}}$.*

Proof. For a Mealy machine to be in $\mathbf{Mealy}_{\mathcal{I}}$, it must have a finite number of states and must be monotone. Lemma 32 shows the latter. For the former, stream functions in $\mathbf{Stream}_{\mathcal{I}}$ have finitely many stream derivatives, so the state set in $\langle f \rangle_{\mathcal{I}}$ is finite. Finally, we must show that composition is preserved: this follows simply because the Mealy function of the resulting machine is defined using the initial output and stream derivative functions. $\square$

4.2 Synthesising a circuit

Although it is easy to recover a Mealy machine from a stream function, recovering a circuit morphism is slightly more subtle. The procedure is standard in circuit design [KJ09]: every state of the Mealy machine is *encoded* as a power of values in $\mathbf{V}$, and then combinational logic is used to transform states and inputs into appropriate outputs and transitions.

In our context, we need to be more careful as the encoding must also respect monotonicity. Recall from Definition 30 that the states in a Mealy machine inherit an ordering from their corresponding stream functions.

Example 48. The state ordering for the Mealy machine in Fig. 3 is $s_0 \preceq \{s_2\}, s_1 \preceq \{s_0, s_2, s_3\}, s_3 \preceq \{s_2\}, s_2 \preceq \{\}$.

Definition 49 (Encoding)**.** *Let S be a state space with an ordering $\preceq$, and let $\mathsf{num}_\gamma\colon S \to \mathbb{N}_{|S|}$ be a bijective function assigning each state a natural number from 0 to $|S| - 1$. Then, the γ-*encoding *of S is a function $\gamma\colon S \to \mathbf{V}^{|S|}$ defined as $\gamma(s)(i) := \top$ if $\mathsf{num}_\gamma^{-1}(i) \preceq s$ and $\gamma(s)(i) := \bot$ otherwise.*

For a given state ordering, there may be multiple encodings depending on how the states are numbered. However, as we shall see later, this does not affect the final result.

Lemma 50. *In the context of Definition 49, $s \preceq s'$ if and only if $\gamma(s) \sqsubseteq \gamma(s')$.*

Proof. First the $(\Rightarrow)$ direction. Let $s_i \preceq s_j$: then $\gamma(s_i)(i) = \top$, $\gamma(s_j)(i) = \bot$, $\gamma(s_i)(j) = \bot$ and $\gamma(s_j)(j) = \top$. We now need to show that for $k \neq i, j$, $\gamma(s_i)(k) \sqsubseteq \gamma(s_j)(k)$. Assume that $\gamma(s_i) \not\sqsubseteq \gamma(s_j)(k)$. Since the tuples can only contain $\bot$ and $\top$, this means that $\gamma(s_i)(k) = \top$ and $\gamma(s_j)(k) = \bot$. By definition of γ, this means that $s_k \leq s_i$ and $s_k \leq s_j$. But this contradicts transitivity as $s_i \preceq s_j$, so the statement holds.

Now the $(\Leftarrow)$ direction. Assume that $\gamma(s_i) \sqsubseteq \gamma(s_j)$. We need to show that $s_i \preceq s_j$. In particular, this means that $\gamma(s_i)(i) \sqsubseteq \gamma(s_j)(i)$. By definition of the encoding γ and reflexivity, $\gamma(s_i)(i) = \top$. Therefore $\gamma(s_j)(i)$ must also be $\top$, as $\gamma(s_j)(i) \leq \gamma(s_j)(i)$. Again by definition of the encoding γ, $\gamma(s_j)(i) = \top$ when $s_i \preceq s_j$. So the statement holds. $\square$

The goal is to construct a combinational circuit morphism that, when interpreted as a function, implements the output and transition function of the Mealy machine. However, such a morphism may not exist for all interpretations.

Definition 51 (Functional completeness)**.** *An interpretation $\mathcal{I} = (\mathbf{V}, [\![-]\!]^{\mathbf{V}}, [\![-]\!]^{\mathbf{G}})$ is* functionally complete *if all monotone functions $\mathbf{V}^m \to \mathbf{V}^n$ are expressible using functions in the image of $[\![-]\!]^{\mathbf{G}}$.*

Example 52. The Belnap functions from Example 11 are functionally complete.

Lemma 53. *For a functionally complete interpretation $\mathcal{I} = (\mathbf{V}, [\![-]\!]^{\mathbf{V}}, [\![-]\!]^{\mathbf{G}})$ and monotone function $f\colon \mathbf{V}^m \to \mathbf{V}^n$, there exists a combinational circuit ${}^{m}\!-\!\boxed{\langle\!\langle f \rangle\!\rangle}\!-\!{}^{n} \in \mathbf{CCirc}_\Sigma$ such that $[\![-\!\boxed{\langle\!\langle f \rangle\!\rangle}\!-]\!]^{\mathbf{C}}_{\mathcal{I}} = f$.*

Proof. Since $\mathcal{I}$ is functionally complete, all functions $\mathbf{V}^m \to \mathbf{V}^n$ are expressible using functions in the image of $[\![-]\!]^{\mathbf{G}}$. Therefore, the circuit $\boxed{\langle\!\langle f \rangle\!\rangle}$ is constructed from the generators in Σ that are mapped to the appropriate functions by $[\![-]\!]^{\mathbf{G}}$. $\square$

To retrieve a circuit from a Mealy machine in a functionally complete interpretation, we just need to define a suitable monotone function. There is an obvious candidate: the Mealy function.

Definition 54 (Monotone completion)**.** *For partial orders M, N, P such that $M \subseteq N$, and a function $f\colon M \to P$, let the* monotone completion *of f be the function $f_{\mathsf{m}}\colon N \to P$ recursively defined as*

$$f_{\mathsf{m}}(v) = \begin{cases} f(v) & \text{if } v \in M \\ \top & \text{if } v = \top, \top \notin M \\ \bigsqcap\{f_{\mathsf{m}}(w) \mid v \sqsubseteq w\} & \text{otherwise} \end{cases}$$

Definition 55 (Mealy encoding)**.** *Given a Mealy machine (S, f, s_0) and an encoding γ, the* Mealy encoding over γ *is a function $\gamma(f)\colon \mathbf{V}^{|S|} \times \mathbf{V}^m \to \mathbf{V}^{|S|} \times \mathbf{V}^n$: the monotone completion of $(\gamma(s), \overline{x}) \mapsto (\gamma(f_T(s, \overline{x})), f_O(s, \overline{x}))$.*

Lemma 56. *Any Mealy encoding is monotone.*

Proof. Any monotone completion of a monotone function is monotone; both γ and γ^{-1} are monotone by Lemma 50; and f is monotone by definition, so the entire function is monotone. $\square$

Definition 57. *For a functionally complete interpretation $\mathcal{I}$, let $\langle\!\langle - \rangle\!\rangle_{\mathcal{I}}\colon \mathbf{Mealy}_{\mathcal{I}} \to \mathbf{SCirc}_{\Sigma}$ be the traced PROP morphism defined for a monotone Mealy machine (S, f, s_0) and encoding γ as* $\boxed{\overline{s}}\,\boxed{\langle\!\langle \gamma(f) \rangle\!\rangle}$, *where $\overline{s} := \gamma(s_0)$.*

As with the translation from circuits to Mealy machines, the reverse direction preserves behaviour. To show this, the notion of state after some transitions must be formalised syntactically.

Definition 58 (Circuit state)**.** *For a sequential circuit* $\boxed{F}$ *with a Mealy form* $\boxed{\overline{s}}\,\boxed{\hat{F}}$, *let its* state at time i for input $\sigma \in (\mathbf{V}^m)^\omega$ *be a sequential circuit* $\boxed{\hat{F}^i_\sigma}$, *defined inductively over i as* $\boxed{\hat{F}^0_\sigma} := \boxed{\overline{s}}$ *and* $\boxed{\hat{F}^{k+1}_\sigma} := \boxed{\sigma(i)}\,\boxed{\hat{F}^i_\sigma}\,\boxed{\hat{F}}$

Lemma 59. *For any sequential circuit* $\boxed{F}$, $\pi_0\left([\![\boxed{F}]\!]^{\mathbf{S}}_{\mathcal{I}}\right) = [\![\boxed{F}]\!]^{\mathbf{S}}_{\mathcal{I}}$, $\pi_1\left([\![\boxed{F}]\!]^{\mathbf{S}}_{\mathcal{I}}\right) = [\![\boxed{F}]\!]^{\mathbf{S}}_{\mathcal{I}}$.

Proof. Immediate by Definition 24. $\square$

Lemma 60. *For a combinational circuit* $\boxed{F}$, *and values $\overline{v} \in \mathbf{V}^m$,* $[\![\boxed{G}\,\boxed{F}]\!]^{\mathbf{S}}_{\mathcal{I}}(0) = [\![\boxed{F}]\!]^{\mathbf{C}}_{\mathcal{I}}([\![\boxed{G}]\!]^{\mathbf{S}}_{\mathcal{I}}(0))$.

Proof. $[\![\boxed{\overline{v}}\,\boxed{F}]\!]^{\mathbf{S}}_{\mathcal{I}}(0) = [\![\boxed{F}]\!]^{\mathbf{S}}_{\mathcal{I}}(\overline{v} :: \bot^\omega)(0) = [\![\boxed{F}]\!]^{\mathbf{C}}_{\mathcal{I}}(\overline{v})$ $\square$

Lemma 61. *Let* $\boxed{F}$ *be a sequential circuit with Mealy form* $\boxed{\overline{s}}\,\boxed{\hat{F}}$ *and define* $f := [\![\boxed{\overline{s}}\,\boxed{\hat{F}}]\!]^{\mathbf{S}}_{\mathcal{I}}$. *Then* $[\![\boxed{\hat{F}^{i+1}_\sigma}]\!]^{\mathbf{S}}_{\mathcal{I}}(0) = \mu_f(\sigma)(i)$ *in the sense of Proposition 21.*

Proof. First note that since μ_f is a fixpoint, $\mu_f(\sigma) = \pi_0\left([\![\boxed{\overline{s}}\,\boxed{\hat{F}}]\!]^{\mathbf{S}}_{\mathcal{I}}(\mu_f(\sigma), \sigma)\right)$. Now we use induction on i. Let $i = 0$:

$$\mu_f(\sigma)(0) = \pi_0\left(\left\llbracket \boxed{\overline{s}}\ \boxed{\hat F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\mu_f(\sigma),\sigma)\right)(0)$$

$$= \pi_0\left(\left\llbracket \boxed{\hat F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\mathsf{shift}(\mu_f(\sigma)) \sqcup \mathsf{val}_{\overline{s}},\sigma)\right)(0) \qquad \text{Definition 24}$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\mathsf{shift}(\mu_f(\sigma)) \sqcup \mathsf{val}_{\overline{s}},\sigma)(0) \qquad \text{Lemma 59}$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}}(\mathsf{shift}(\mu_f(\sigma))(0) \sqcup \mathsf{val}_{\overline{s}}(0),\sigma(0)) \qquad \text{Definition 24}$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}}(\bot \sqcup \overline{s},\sigma(0)) \qquad \text{Definition 22}$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}}(\overline{s},\sigma(0)) \qquad \bot \sqcup x = x$$

$$= \left\llbracket \boxed{\sigma(0)}\ \boxed{\overline{s}}\ \boxed{\hat F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(0) \qquad \text{Lemma 60}$$

$$= \left\llbracket \boxed{\sigma(0)}\ \boxed{\hat F^0_\sigma}\ \boxed{\hat F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(0) \qquad \text{Definition 58}$$

$$= \left\llbracket \boxed{\hat F^1_\sigma} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(0) \qquad \text{Definition 58}$$

Now $i = k+1$.

$$\mu_f(\sigma)(k+1) = \pi_0\left(\left\llbracket \boxed{\overline{s}}\ \boxed{\hat F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\mu_f(\sigma),\sigma)\right)(k+1)$$

$$= \pi_0\left(\left\llbracket \boxed{\hat F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\mathsf{shift}(\mu_f(\sigma)) \sqcup \mathsf{val}_{\overline{s}},\sigma)\right)(k+1) \qquad \text{Definition 24}$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\mathsf{shift}(\mu_f(\sigma)) \sqcup \mathsf{val}_{\overline{s}},\sigma)(k+1) \qquad \text{Lemma 59}$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}}(\mathsf{shift}(\mu_f(\sigma))(k+1) \sqcup \mathsf{val}_{\overline{s}}(k+1),\sigma(k+1)) \qquad \text{Definition 24}$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}}(\mu_f(\sigma)(k) \sqcup \bot,\sigma(k+1)) \qquad \text{Definition 22}$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}}(\mu_f(\sigma)(k),\sigma(k+1)) \qquad x \sqcup \bot = x$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}}\left(\left\llbracket \boxed{\hat F^{k+1}_\sigma} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(0),\sigma(k+1)\right) \qquad \text{IH}$$

$$= \left\llbracket \boxed{\sigma(k+1)}\ \boxed{\hat F^{k+1}_\sigma}\ \boxed{\hat F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(0) \qquad \text{Lemma 60}$$

$$= \left\llbracket \boxed{\hat F^{k+2}_\sigma} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(0) \qquad \text{Definition 58}$$

$\square$

Lemma 62. *Let* $\boxed{F}$ *be defined as* $\left\llbracket \boxed{\overline{s}}\ \boxed{\hat F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}$. *Then* $\left\llbracket \boxed{F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)(i) = \left\llbracket \boxed{\sigma(i)}\ \boxed{\hat F^i_\sigma}\ \boxed{\hat F} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}}(0)$.

Proof. First we apply various definitions:

$$\left\llbracket \boxed{F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)(i) = \left\llbracket \boxed{\overline{s}}\ \boxed{\hat F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}$$

$$= \pi_1\left(\left\llbracket \boxed{\overline{s}}\ \boxed{\hat F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\mu(\sigma),\sigma)\right)(i) \qquad \text{Proposition 21}$$

$$= \pi_1\left(\left\llbracket \boxed{F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\mathsf{shift}(\mu(\sigma)) \sqcup \mathsf{val}_v,\sigma)\right)(i) \qquad \text{Definition 24}$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\mathsf{shift}(\mu(\sigma)) \sqcup \mathsf{val}_v,\sigma)(i) \qquad \text{Lemma 59}$$

$$= \left\llbracket \boxed{F} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}}(\mathsf{shift}(\mu(\sigma))(i) \sqcup \mathsf{val}_{\overline{s}}(i),\sigma(i)) \qquad \text{Definition 24}$$

Now we perform induction on i. For $i = 0$:

$$
\begin{aligned}
\left\llbracket {}_{m}^{x}\!\boxed{F}\!\!-_{n} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} (\mathsf{shift}(\mu(\sigma))(0) \sqcup \mathsf{val}_{\overline{s}}(0), \sigma(0)) &= \left\llbracket {}_{m}^{x}\!\boxed{F}\!\!-_{n} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} (\bot \sqcup \overline{s}, \sigma(0)) && \text{Definition 22} \\
&= \left\llbracket {}_{m}^{x}\!\boxed{F}\!\!-_{n} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} (\overline{s}, \sigma(0)) \\
&= \left\llbracket \boxed{\sigma(0)}\ \boxed{\overline{s}}\ \boxed{\hat{F}}\!\!-_{n} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} (0) && \text{Lemma 60} \\
&= \left\llbracket \boxed{\sigma(0)}\ \boxed{\hat{F}^{0}_{\sigma}}\ \boxed{\hat{F}}\!\!-_{n} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} (0) && \text{Definition 58}
\end{aligned}
$$

Now for $i = k + 1$.

$$
\begin{aligned}
&\left\llbracket {}_{m}^{x}\!\boxed{F}\!\!-_{n} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} (\mathsf{shift}(\mu(\sigma))(k+1) \sqcup \mathsf{val}_{\overline{s}}(k+1), \sigma(k+1)) \\
&= \left\llbracket {}_{m}^{x}\!\boxed{F}\!\!-_{n} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} (\mu(\sigma)(k) \sqcup \bot, \sigma(k+1)) && \text{Definition 22} \\
&= \left\llbracket {}_{m}^{x}\!\boxed{F}\!\!-_{n} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} (\mu(\sigma)(k), \sigma(k+1)) && x \sqcup \bot = x \\
&= \left\llbracket {}_{m}^{x}\!\boxed{F}\!\!-_{n} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} \left(\left\llbracket \boxed{\hat{F}^{i+1}_{\sigma}}\!\!-_{x} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} (0), \sigma(k+1)\right) && \text{IH} \\
&= \left\llbracket \boxed{\sigma(k+1)}\ \boxed{\hat{F}^{k+1}_{\sigma}}\ \boxed{\hat{F}}\!\!-_{n} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} (0) && \text{Lemma 60}
\end{aligned}
$$

□

Lemma 63. *For a Mealy machine (S, f) with initial state s and encoding γ, let $\boxed{\hat{F}^{i}_{\sigma}}\!\!-_{x}$ be the circuit state of $\left\llbracket {}_{m}\!\!-\boxed{\overline{s}}\ \boxed{\langle\!\langle \gamma(f) \rangle\!\rangle}\!\!-_{n} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}$, where $\overline{s} = \gamma(s)$. Then $\left\llbracket \boxed{\hat{F}^{i}_{\sigma}}\!\!-_{x} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} (0) = \gamma(T_{f,i}(s, \sigma))$.*

Proof. We use induction on i.

$$
\begin{aligned}
\left\llbracket \boxed{\hat{F}^{0}_{\sigma}}\!\!-_{x} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} (0) &= \left\llbracket \boxed{\overline{s}}\!- \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} (0) && \text{Definition 58} \\
&= \gamma(s) && \text{Definition 24} \\
&= \gamma(T_{f,0}(s, \sigma)) && \text{Definition 27}
\end{aligned}
$$

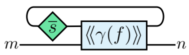
Now $i = k + 1$:

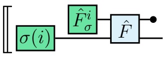
$$
\begin{aligned}
\left\llbracket \boxed{\hat{F}^{k+1}_{\sigma}}\!- \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} (0) &= \left\llbracket \boxed{\sigma(k+1)}\ \boxed{\hat{F}^{k}_{\sigma}}\ \boxed{\langle\!\langle \gamma(f) \rangle\!\rangle}\!\!- \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} (0) && \text{Definition 58} \\
&= \pi_0 \left(\left\llbracket \boxed{\sigma(k+1)}\ \boxed{\hat{F}^{k}_{\sigma}}\ \boxed{\langle\!\langle \gamma(f) \rangle\!\rangle}\!\!- \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} (0) \right) && \text{Lemma 59} \\
&= \pi_0 \left(\left\llbracket {}_{m}^{x}\!\boxed{\langle\!\langle \gamma(f) \rangle\!\rangle}\!\!-_{n}^{x} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} \left(\left\llbracket \boxed{\hat{F}^{k}_{\sigma}}\!- \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} (0), \sigma(k+1) \right) \right) && \text{Lemma 60} \\
&= \pi_0 \left(\left\llbracket {}_{m}^{x}\!\boxed{\langle\!\langle \gamma(f) \rangle\!\rangle}\!\!-_{n}^{x} \right\rrbracket^{\mathbf{C}}_{\mathcal{I}} (\gamma(T_{f,k}(s, \sigma)), \sigma(k+1)) \right) && \text{IH} \\
&= \pi_0 \left(\gamma(f) \left(\gamma(T_{f,k}(s, \sigma)), \sigma(k+1) \right) \right) && \text{Lemma 53} \\
&= \gamma \left(f_T \left(\gamma^{-1}(\gamma(T_{f,k}(s, \sigma))), \sigma(k+1) \right) \right) && \text{Definition 55} \\
&= \gamma \left(f_T \left(T_{f,k}(s, \sigma), \sigma(k+1) \right) \right) && \text{Inverse} \\
&= \gamma \left(T_{f,k+1}(s, \sigma) \right) && \text{Definition 27}
\end{aligned}
$$

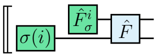
□

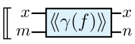
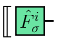
Theorem 64. $!(-) = \llbracket - \rrbracket^{\mathbf{S}}_{\mathcal{I}} \circ \langle\!\langle - \rangle\!\rangle_{\mathcal{I}}$.

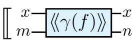
Proof. Recall that, for any $(S, f, s) \in \mathbf{Mealy}_{\mathcal{I}}$, $!(S, f, s)(\sigma)(i) := f_O(T_{f,i}(s, \sigma), \sigma(i))$ by Proposition 28. So we need to show that $[\![\langle\!\langle (S, f, s) \rangle\!\rangle_{\mathcal{I}}]\!]^{\mathbf{S}}_{\mathcal{I}} = f_O(T_{f,i}(s, \sigma), \sigma(i))$.

$[\![\langle\!\langle (S, f, s) \rangle\!\rangle_{\mathcal{I}}]\!]^{\mathbf{S}}_{\mathcal{I}} (\sigma)(i) = \Big[\!\!\Big[$  $\Big]\!\!\Big]^{\mathbf{S}}_{\mathcal{I}} (\sigma)(i)$ — Definition 57

$= \Big[\!\!\Big[$  $\Big]\!\!\Big]^{\mathbf{S}}_{\mathcal{I}}$ — Lemma 62

$= \pi_1 \Big(\Big[\!\!\Big[$  $\Big]\!\!\Big]^{\mathbf{S}}_{\mathcal{I}} \Big)$ — Lemma 59

$= \pi_1 \Big(\Big[\!\!\Big[$  $\Big]\!\!\Big]^{\mathbf{C}}_{\mathcal{I}} \Big(\Big[\!\!\Big[$  $\Big]\!\!\Big]^{\mathbf{S}}_{\mathcal{I}}, \sigma(i) \Big) \Big)$ — Lemma 60

$= \pi_1 \Big(\Big[\!\!\Big[$  $\Big]\!\!\Big]^{\mathbf{C}}_{\mathcal{I}} (\gamma(T_{f,i}(s, \sigma)), \sigma(i)) \Big)$ — Lemma 63

$$
\begin{aligned}
&= \pi_1 \left(\gamma(f) \left(\gamma(T_{f,i}(s, \sigma)), \sigma(i) \right) \right) && \text{Lemma 53} \\
&= f_O \left(\gamma^{-1} \left(\gamma(T_{f,i}(s, \sigma)) \right), \sigma(i) \right) && \text{Definition 55} \\
&= f_O \left(T_{f,i}(s, \sigma), \sigma(i) \right) && \text{Inverse}
\end{aligned}
$$

$\square$

This means that each function in $\mathbf{Stream}_{\mathcal{I}}$ has at least one circuit in $\mathbf{SCirc}_{\Sigma}$ with the same behaviour under $\mathcal{I}$.

Theorem 65. *Define two PROP morphisms*

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

Then the following statements hold:

$$
\begin{aligned}
[\![-]\!]^{\mathbf{S}}_{\mathcal{I}} \circ \psi \circ \phi &= [\![-]\!]^{\mathbf{S}}_{\mathcal{I}} \\
\phi \circ \psi &= \mathsf{id}_{\mathbf{Stream}_{\mathcal{I}}}
\end{aligned}
$$

Proof. First we show the former:

$$
\begin{aligned}
[\![-]\!]^{\mathbf{S}}_{\mathcal{I}} \circ \psi \circ \phi &= [\![-]\!]^{\mathbf{S}}_{\mathcal{I}} \circ \langle\!\langle - \rangle\!\rangle_{\mathcal{I}} \circ \langle - \rangle_{\mathcal{I}} \circ !(-) \circ [-]_{\mathcal{I}} \\
&= \, !(-) \circ \langle - \rangle_{\mathcal{I}} \circ !(-) \circ [-]_{\mathcal{I}} && \text{Theorem 64} \\
&= \, !(-) \circ [-]_{\mathcal{I}} && \text{Corollary 44} \\
&= [\![-]\!]^{\mathbf{S}}_{\mathcal{I}} && \text{Theorem 64}
\end{aligned}
$$

Now the latter:

$$
\begin{aligned}
\phi \circ \psi &= \, !(-) \circ [-]_{\mathcal{I}} \circ \langle\!\langle - \rangle\!\rangle_{\mathcal{I}} \circ \langle - \rangle_{\mathcal{I}} \\
&= [\![-]\!]^{\mathbf{S}}_{\mathcal{I}} \circ \langle\!\langle - \rangle\!\rangle_{\mathcal{I}} \circ \langle - \rangle_{\mathcal{I}} && \text{Theorem 43} \\
&= \, !(-) \circ \langle - \rangle_{\mathcal{I}} && \text{Theorem 64} \\
&= \mathsf{id}_{\mathbf{Stream}_{\mathcal{I}}} && \text{Corollary 44}
\end{aligned}
$$

$\square$

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.

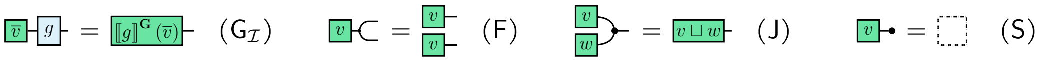


Figure 4: Axioms of $\mathbf{SCirc}^{\mathcal{C}}_{\Sigma,\mathcal{I}}$.

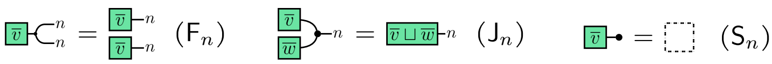


Figure 5: Generalisations of the equations in Fig. 4.

Definition 66 (Extensional equivalence)**.** *Two sequential circuits* $m\text{-}\boxed{F}\text{-}n$ *and* $m\text{-}\boxed{G}\text{-}n$ *are* extensionally equivalent *if* $[\![\text{-}\boxed{F}\text{-}]\!]^{\mathbf{S}}_{\mathcal{I}} = [\![\text{-}\boxed{G}\text{-}]\!]^{\mathbf{S}}_{\mathcal{I}}$.

As the stream functions in $\mathbf{Stream}_{\mathcal{I}}$ have finitely many stream derivatives, to check if two streams are equal we only need to check that the outputs of the function are equal for a certain number of elements.

Proposition 67 ([GJL17b])**.** *Two sequential circuits* $m\text{-}\boxed{F}\text{-}n$ *and* $m\text{-}\boxed{G}\text{-}n$ *containing no more than* k *delay generators are extensionally equivalent if and only if* $[\![\text{-}\boxed{F}\text{-}]\!]^{\mathbf{S}}_{\mathcal{I}}(\sigma)(i) = [\![\text{-}\boxed{G}\text{-}]\!]^{\mathbf{S}}_{\mathcal{I}}(\sigma)(i)$ *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. However, rather than checking if two large circuits are equal by comparing their output streams up to some element, it can be more intuitive to reason using equations.

Because $\mathbf{SCirc}_\Sigma$ is a STMC, the equations in Fig. 1 already hold 'by default'. In fact, because we are using string diagrams, these equations are absorbed into the notation and can be 'applied' by moving boxes around while retaining their connectivity. However, to change the components of the circuit, equations will need to be applied explicitly.

Definition 68. *We say that an axiom* $m\text{-}\boxed{F}\text{-}n = m\text{-}\boxed{G}\text{-}n$ *is* valid for $\mathcal{I}$ *if* $[\![m\text{-}\boxed{F}\text{-}n]\!]^{\mathbf{S}}_{\mathcal{I}} = [\![m\text{-}\boxed{G}\text{-}n]\!]^{\mathbf{S}}_{\mathcal{I}}$.

5.1 Combinational axioms

Let us first provide a set of axioms such that the result of quotienting $\mathbf{SCirc}_\Sigma$ by this set is as expressive as $\mathbf{Stream}_{\mathcal{I}}$: i.e. if two circuits have the same semantics as streams, then they are also equal in this quotient category. We begin by defining axioms for combinational circuits.

Definition 69 (Combinational axioms)**.** *Let the set* $\mathcal{C}_{\mathcal{I}}$ *of* combinational axioms *be defined asthe equations in Fig. 4*

Lemma 70. *The axioms in* $\mathcal{C}_{\mathcal{I}}$ *are valid for any* $\mathcal{I}$.

Proof. Immediate by examining the stream interpretations. □

Axioms in $\mathcal{C}_{\mathcal{I}}$ show how the structural generators interact with values. The gate ($\mathsf{G}_{\mathcal{I}}$) applies the corresponding function, the fork (F) copies values, the join (J) coalesces two values using the join operation $\sqcup$, and the stub (S) discards values. These axioms also generalise to the 'composite' values by axioms of STMCs.

Definition 71. *Let* $\mathbf{SCirc}^{\mathcal{C}}_{\Sigma,\mathcal{I}}$ *be defined as* $\mathbf{SCirc}_\Sigma/\mathcal{C}_{\mathcal{I}}$.

Lemma 72. *For any values* $\overline{v}, \overline{w} \in \mathbf{V}^m$*, the equations in Fig. 5 hold in* $\mathbf{SCirc}^{\mathcal{C}}_{\Sigma,\mathcal{I}}$.

Proof. Immediate by Notation 5 and applying the appropriate equation in $\mathcal{C}_{\mathcal{I}}$ multiple times. □

Theorem 73 (Extensionality)**.** *For any combinational circuit* $m\text{-}\boxed{F}\text{-}n$ *and values* $\overline{v} \in \mathbf{V}^m$ *there exist values* $\overline{w} \in \mathbf{V}^n$ *such that* $\boxed{\overline{v}}\text{-}\boxed{F}\text{-}n = \boxed{\overline{w}}\text{-}$ *in* $\mathbf{SCirc}^{\mathcal{C}}_{\Sigma,\mathcal{I}}$.

Proof. By induction over the structure of $\text{-}\boxed{F}\text{-}$: the statement holds for the generators in $\mathbf{SCirc}_\Sigma$ by using the appropriate equation in $\mathcal{C}_{\mathcal{I}}$. The inductive cases are trivial. □

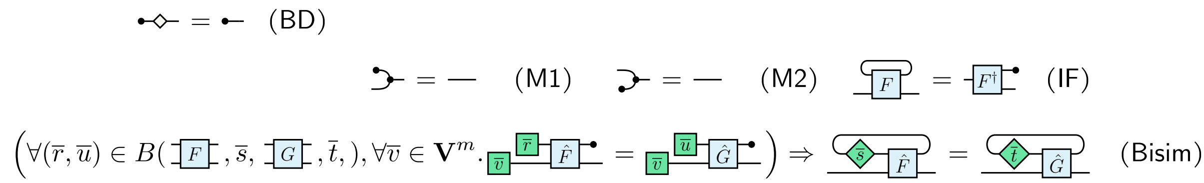
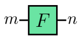
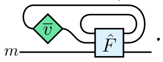


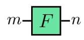
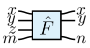
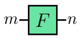
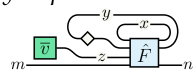
Figure 6: Axioms of $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$, in addition to those in Fig. 4.

5.2 Non-delay-guarded feedback

The equations in $\mathcal{C}_{\mathcal{I}}$ suffice for reasoning with closed combinational circuits, but generally circuits are neither closed nor combinational. A general equation is needed that translates between open circuits with the same input-output behaviour using the combinational equations. From this equation, more useful fine-grained equations can be derived.

Rather than reasoning with arbitrary sequential circuits it is useful to isolate the combinational and the sequential components.

Definition 74 (Pre-Mealy form)**.** *A sequential circuit*  *is said to be in* pre-Mealy form *if it is in the form* .

Lemma 75 (Global trace-delay form)**.** *For any sequential circuit*  *there exists combinational circuit*  *and values $\overline{v} \in \mathbf{V}^z$ such that*  $=$  *by axioms of STMCs.*

Proof. A trace can be inserted before any delay generator using (Yanking). Then both these and the pre-existing traces can be transformed into 'global traces' by applying (Tightening) and (Superposing), which yields the desired form. □

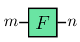
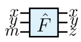
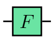
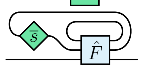
Lemma 76. (BD), (M1) *and* (M2) *in Fig. 6 are valid for any $\mathcal{I}$.*

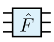
Proof. It is a simple exercise to check the corresponding stream functions. □

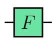
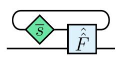
Quotienting by these axioms yields a category in which any circuit is equal to one in pre-Mealy form.

Definition 77. *Let $\mathbf{SCirc}^{\mathcal{M}}_{\Sigma,\mathcal{I}}$ be defined as*

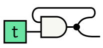
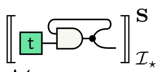
$$\mathbf{SCirc}^{\mathcal{C}}_{\Sigma,\mathcal{I}}/((\text{BD}) + (\text{M1}) + (\text{M2}))$$

Lemma 78. *For a sequential circuit*  *there exists at least one combinational circuit*  *and tuple of values $\overline{s} \in \mathbf{V}^x$ such that*  $=$  *in $\mathbf{SCirc}^{\mathcal{M}}_{\Sigma,\mathcal{I}}$.*

We call the circuit  a *combinational core*. The reasoning behind the name 'pre-Mealy form' is clear when one compares the form with the result of applying $\langle\!\langle - \rangle\!\rangle_{\mathcal{I}}$ from Definition 57 to a Mealy machine: it is almost the same other than the addition of the non-delay-guarded trace.

Definition 79 (Mealy form)**.** *A sequential circuit*  *is said to be in* Mealy form *if it is in the form*  *i.e. all feedback passes through a delay.*

Although all circuits in $\mathbf{SCirc}^{\mathcal{M}}_{\Sigma,\mathcal{I}}$ are equal to a circuit in pre-Mealy form, the same is not true for Mealy form.

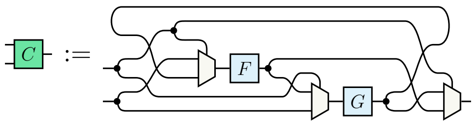
Example 80. Consider the circuit . In the stream semantics,  $()(i) = \bot$ for all $i \in \mathbb{N}$. However, this circuit is not equal to a circuit in Mealy form in $\mathbf{SCirc}^{\mathcal{M}}_{\Sigma,\mathcal{I}}$.

In circuit design, it is common to enforce that circuits have no non-delay-guarded feedback in order to avoid undefined behaviour. One might ask if the delay-guarded feedback condition should enforced in our framework too in order to stick to 'well-behaved' circuits. However, careful use of non-delay-guarded feedback can still result in circuits that have useful output: for example, it can be used as a clever way of sharing resources [Rie04].

Example 81. We extend Σ_B with a new gate symbol MUX for a *multiplexer* with $\#(\mathrm{MUX}) = 3$. This is drawn as . The interpretation of the multiplexer is

$$\llbracket - \rrbracket^{\mathbf{G}}(\mathrm{MUX})(\mathsf{f}, x, y) = x \qquad \llbracket - \rrbracket^{\mathbf{G}}(\mathrm{MUX})(\bot, x, y) = \bot \wedge (x \vee y)$$
$$\llbracket - \rrbracket^{\mathbf{G}}(\mathrm{MUX})(\mathsf{t}, x, y) = y \qquad \llbracket - \rrbracket^{\mathbf{G}}(\mathrm{MUX})(\top, x, y) = \top \wedge (x \vee y)$$

Observe the following circuit, from [MSB12, Fig. 1], where F and G are arbitrary combinational circuits.



This circuit is already in pre-Mealy form, and has non-delay-guarded feedback. Despite this, it produces useful output when the control signal is 0 or 1:

$$\llbracket F \rrbracket^{\mathbf{S}}_{\mathcal{I}_\star}(\sigma)(i) = \llbracket F \, G \rrbracket^{\mathbf{C}}_{\mathcal{I}_\star}(\pi_2(\sigma(i))) \text{ if } \pi_0(\sigma(i)) = 0$$
$$\llbracket F \rrbracket^{\mathbf{S}}_{\mathcal{I}_\star}(\sigma)(i) = \llbracket G \, F \rrbracket^{\mathbf{C}}_{\mathcal{I}_\star}(\pi_2(\sigma(i))) \text{ if } \pi_0(\sigma(i)) = 1$$

An equation is required to eliminate the 'instant feedback'. We turn to the Kleene fixed-point theorem.

Lemma 82. *For a monotone function $f\colon \mathbf{V}^{n+m} \to \mathbf{V}^n$ and $i \in \mathbb{N}$, let $f_i\colon \mathbf{V}^m \to \mathbf{V}^n$ be defined as $f^0(x) = f(\bot^n, x)$ 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. □

Definition 83 (Iteration). *For a combinational circuit ${}_m^x F {}_n^x$, let its nth* iteration *${}_m F^n {}_n^x$ be defined inductively as $F^0 :=$ $\bullet F$ and $F^{k+1} :=$ F^k F.*

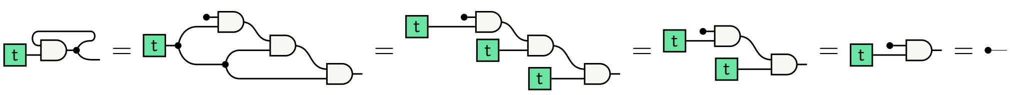
Definition 84 (Fixpoint). *Given an interpretation with value lattice $\mathbf{V}$, the* fixpoint *of a combinational circuit ${}_m^x F {}_n^x$, denoted as ${}_m F^\dagger {}_n^x$, is defined as F^{c+1} where c is the length of the longest chain in $\mathbf{V}^x$.*

Proposition 85 (Instant feedback equation). *The* (IF) *equation in Fig. 6 is valid for any $\mathcal{I}$.*

Proof. By applying Lemma 82 pointwise. □

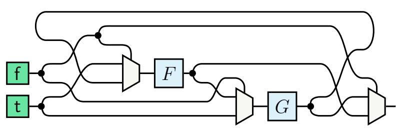
If applied locally for every feedback loop, the (IF) equation would cause an exponential blowup of the circuit. However, if a circuit is in global trace-delay form, the (IF) equation need only be applied once to the 'global feedback loop'. Although the value of c increases as the number of feedback wires increases, it only does so linearly as it depends on the height of the lattice rather than the total number of elements.

Example 86. Recall Example 80. In $\mathbf{SCirc}^{\mathcal{M}}_{\Sigma,\mathcal{I}}$, the trace is unfolded by (IF):

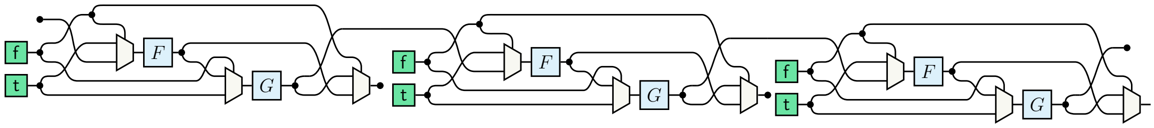


The combinational equations reduce this to the $\bullet$ value.

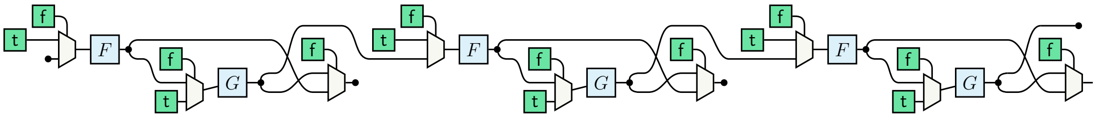
Example 87. Recall the cyclic combinational circuit from Example 81. When applied to some inputs, this can also be reduced appropriately using (IF). We will precompose the circuit with values so it only produces interesting output on the first tick, and then reduce it using equations in $\mathcal{C}_{\mathcal{I}}$.



In particular, note that the control switch is set to f. We then apply (IF) to eliminate the feedback loop.



Using (F), the values can be propagated across the circuit.



It is a simple exercise to reduce the resulting circuit to $\boxed{\mathsf{t}}$-$\boxed{F}$-$\boxed{G}$- by applying ($\mathsf{G}_{\mathcal{I}}$), (F) and (S). When the control switch is set to t the procedure is similar, but results in the two combinational circuits being applied in reverse. This corresponds exactly with the stream semantics stated in Example 81.

Definition 88. *Let* $\mathbf{SCirc}^{\mathcal{D}}_{\Sigma,\mathcal{I}} := \mathbf{SCirc}^{\mathcal{M}}_{\Sigma,\mathcal{I}}/(\mathsf{IF})$.

$\mathbf{SCirc}^{\mathcal{D}}_{\Sigma,\mathcal{I}}$ is a category in which every circuit with non-delay-guarded feedback is equal to a circuit in Mealy form, by using (IF).

Theorem 89. *For any sequential circuit* $m\text{-}\boxed{F}\text{-}n$, *there exist at least one combinational circuit* ${}^{x}_{m}\boxed{\hat{F}}^{x}_{n}$ *and values* $s \in \mathbf{V}^x$ *such that* $m\text{-}\boxed{F}\text{-}n = {}_{m}\boxed{\langle\bar{s}\rangle\ \hat{F}}_{n}$ *in* $\mathbf{SCirc}^{\mathcal{D}}_{\Sigma,\mathcal{I}}$.

Proof. In $\mathbf{SCirc}^{\mathcal{M}}_{\Sigma,\mathcal{I}}$, any sequential circuit is equal to a circuit in pre-Mealy form by Lemma 78. Then, since the core is a combinational circuit with a non-delay-guarded trace, it is equal to a circuit without a non-delay-guarded trace by (IF). □

5.3 Bisimilarity

We now have a setting in which any circuit is equal to a delay-guarded one. To finish a complete equational theory, we require an equation between delay-guarded circuits with the same behaviour. We take inspiration from the concept of *bisimilarity*, which applies to Mealy machines and is defined coinductively.

Definition 90 (Bisimilar states [BRS08])**.** *Given two Mealy machines* (S, f, s_0) *and* (T, g, t_0), $(s, t) \in S \times T$ *are* bisimilar states *if* $f_O(s, a) = g_O(t, a)$ *and* $(f_T(s, a), g_T(t, a))$ *are bisimilar states.* (S, f, s_0) *and* (T, g, t_0) *are* bisimilar *if* (s_0, t_0) *are bisimilar states.*

Bisimilarity can be expressed syntactically, using the notion of circuit state from Definition 58.

Proposition 91. *For any two delay-guarded sequential circuits* $m\text{-}\boxed{F}\text{-}n$ *and* $m\text{-}\boxed{G}\text{-}n$ *with corresponding Mealy forms* $\boxed{\langle\bar{s}\rangle\ \hat{F}}$ *and* $\boxed{\langle\bar{t}\rangle\ \hat{G}}$, $[\![\text{-}\boxed{F}\text{-}]\!]^{\mathbf{S}}_{\mathcal{I}} = [\![\text{-}\boxed{G}\text{-}]\!]^{\mathbf{S}}_{\mathcal{I}}$ *if and only if* $\left[\!\left[\boxed{\sigma(i)}\ \boxed{F^i_\sigma}\ \boxed{\hat{F}}\right]\!\right]^{\mathbf{S}}_{\mathcal{I}} = \left[\!\left[\boxed{\sigma(i)}\ \boxed{G^i_\sigma}\ \boxed{\hat{G}}\right]\!\right]^{\mathbf{S}}_{\mathcal{I}}$ *for all* $\sigma \in (\mathbf{V}^m)^\omega$ *and* $i \in \mathbb{N}$.

Proof. This follows from Lemma 62. □

So the problem of checking if two circuits are equal is reduced to checking if each core produces the same outputs for each circuit state and input stream. Of course, there are infinitely many streams σ, but since these circuits are *finite*, there will only be a finite number of circuit states to compare.

Definition 92 (Circuit bisimulation)**.** *Let* ${}_{m}\boxed{\langle\bar{s}\rangle\ \hat{F}}^{x}_{n}$ *and* ${}_{m}\boxed{\langle\bar{t}\rangle\ \hat{G}}^{y}_{n}$ *be sequential circuits. Then the* circuit bisimulation *of* $\boxed{\hat{F}}$ *and* $\boxed{\hat{G}}$ *from* $(\bar{s}, \bar{t})$ *is a set* $B(\boxed{\hat{F}}, \bar{s}, \boxed{\hat{G}}, \bar{t}) \subseteq \mathbf{V}^x \times \mathbf{V}^y$ *computed as* $\{(\bar{r}, \bar{u}) \mid \sigma \in (\mathbf{V}^m)^\omega, i \in \mathbb{N}, \boxed{F^i_\sigma}\text{-} = \boxed{\bar{r}}\text{-}, \boxed{G^i_\sigma}\text{-} = \boxed{\bar{u}}\text{-}\}$.

Lemma 93. *Any circuit bisimulation is finite.*

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

Theorem 94. $[\![\text{-}\boxed{F}\text{-}]\!]^{\mathbf{S}}_{\mathcal{I}} = [\![\text{-}\boxed{G}\text{-}]\!]^{\mathbf{S}}_{\mathcal{I}}$ *if and only if* $\boxed{v}\ \boxed{\bar{r}}\ \boxed{\hat{F}} = \boxed{v}\ \boxed{\bar{u}}\ \boxed{\hat{G}}$ *by equations in* $\mathcal{C}_{\mathcal{I}}$, *for all* $(\bar{r}, \bar{u}) \in B(\boxed{\hat{F}}, \bar{s}, \boxed{\hat{G}}, \bar{t})$ *and* $v \in \mathbf{V}^m$.

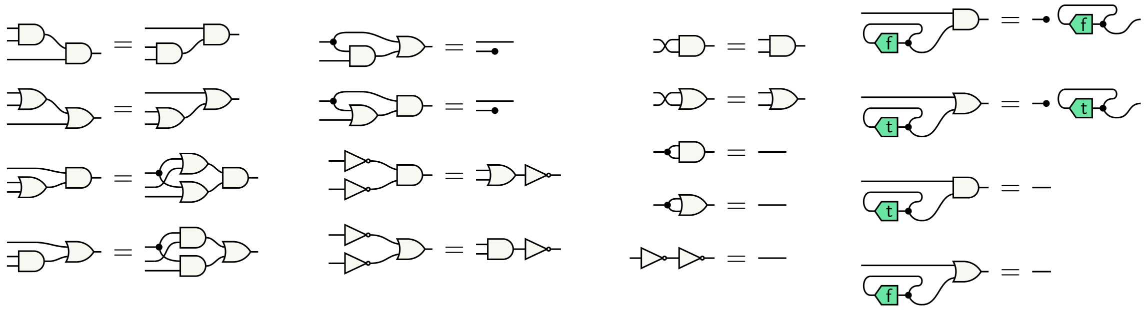


Figure 7: Derivable equations for circuits in $\mathcal{I}_B$.

Proof. This follows by Proposition 91 since each $(\boxed{\bar{F}}, \boxed{\bar{u}}) = (\boxed{\hat{F}^i_\sigma}, \boxed{\hat{G}^i_\sigma})$ by equations in $\mathcal{C}_\mathcal{I}$. □

Corollary 95 (Bisimulation equation)**.** *The* (Bisim) *equation in Fig. 6 is valid for any $\mathcal{I}$.*

Definition 96. *Let* $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}} := \mathbf{SCirc}^{\mathcal{D}}_{\Sigma,\mathcal{I}}/(\text{Bisim})$

$\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$ is a category in which the equational theory is *sound and complete*: if circuits are extensionally equivalent, then they are equal.

Theorem 97. $m\!-\!\boxed{F}\!-\!n = m\!-\!\boxed{G}\!-\!n$ *in* $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$ *if and only if* $[\![-\boxed{F}-]\!]^{\mathbf{S}}_{\mathcal{I}} = [\![-\boxed{G}-]\!]^{\mathbf{S}}_{\mathcal{I}}$.

Proof. The ($\Rightarrow$) direction follows as all equations are valid if and only if $[\![-\boxed{F}-]\!]^{\mathbf{S}}_{\mathcal{I}} = [\![-\boxed{G}-]\!]^{\mathbf{S}}_{\mathcal{I}}$. For the ($\Leftarrow$) direction, first recall that in $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$, any sequential circuit is equal to a circuit in Mealy form by Theorem 89. Then, as $[\![-\boxed{F}-]\!]^{\mathbf{S}}_{\mathcal{I}} = [\![-\boxed{G}-]\!]^{\mathbf{S}}_{\mathcal{I}}$, the preconditions of (Bisim) hold by Theorem 94. This means that there is a sequence of equations in $\mathcal{C}_\mathcal{I}$ + (IF) + (Bisim) that such that $-\boxed{F}- = -\boxed{G}-$. □

6 Local reasoning

The results of the previous sections mean that the axioms of $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$ are enough to capture the semantics of digital circuits. Unfortunately, reasoning in this manner can still be tedious, as the (Bisim) axiom may have many states to check. It is more useful to use this axiom to derive simpler *local* equations: equations that describe the interaction between individual generators. For example, the equations in Fig. 7 hold in $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$ for the interpretation $\mathcal{I}_B$.

However, in this section we will show that for some common circuit tasks it is not necessary to use the 'global' (Bisim) equation at all. Instead one can reason almost entirely with 'local' equations, which is intuitive and also more suitable for operational semantics.

In order to avoid confusion, we will now quotient the original syntactic category $\mathbf{SCirc}_\Sigma$ with a different set of axioms. To identify suitable axioms, we will show that they arise as equations in $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$, thus using the results of the previous section to inform the current one.

6.1 Algebraic structure

The generators (, , ,) form a *bialgebra*.

Definition 98 (Bialgebra equations)**.** *Let $\mathcal{B}$ be the set of* bialgebra *equations, defined as inFig. 8.*

Lemma 99. *The equations in $\mathcal{B}$ hold in* $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$.

Proof. The equations are all combinational so there is only one circuit state to consider. □

Remark 100. The equations in $\mathcal{B}$ extend to the arbitrary width versions from Notation 5 by applying axioms of STMCs and the corresponding single wire equation. We will use the same name for the equation regardless of the width of the wires.

The bialgebra equations will form the basis of our local equational framework. However, there are other interactions between generators in $\mathbf{SCirc}_\Sigma$ that must be captured.

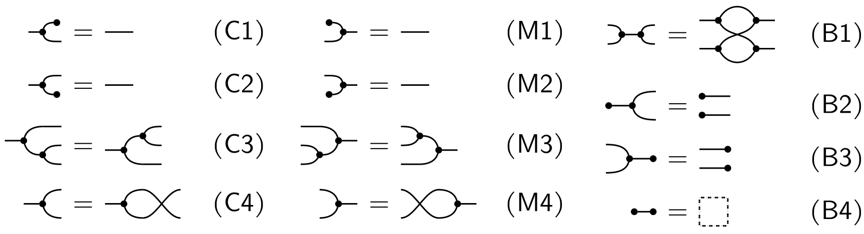


Figure 8: Equations that hold in any bialgebra.

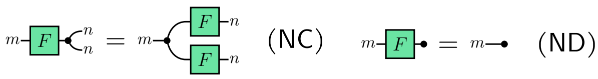


Figure 9: Equations that hold in any Cartesian category.

Definition 101 (Local equations)**.** *Let $\mathcal{L}$ be the set of* local equations, *defined as the equations in Fig. 10.*

Lemma 102. *The equations in $\mathcal{L}$ hold in $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$.*

Proof. In this case it is easier to check the stream functions of each equation and confirm they are equal. □

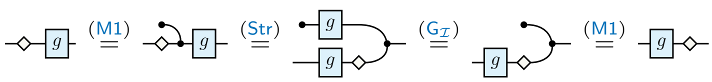
The equations in $\mathcal{L}$ tell us how the generators interact with each other. Most are fairly intuitive; the (Str) equation ('streaming') may need some explanation. The equation tells us that when an input to a gate has an immediate component and a delayed component, it is the same as copying the gate so one copy handles what is happening 'now' and the other handles what is happening 'later'. Intuitively, this says that the join generator is 'almost' a natural transformation. In general this is not the case, as $\llbracket \cdot \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma,\tau)(i) = \llbracket \cdot \rrbracket^{\mathbf{C}}_{\mathcal{I}}(\sigma(i) \sqcup \tau(i))$ and $\llbracket \cdot \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma,\tau)(i) = \llbracket \cdot \rrbracket^{\mathbf{C}}_{\mathcal{I}}(\sigma(i)) \sqcup \llbracket \cdot \rrbracket^{\mathbf{C}}_{\mathcal{I}}(\tau(i))$, which are not necessarily equal. 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.

From a practical point of view, equation (Str) models *retiming* [LS91]: moving registers forward or backward across gates. Forward retiming (left to right in (Str)) is always possible but for backward retiming (right to left in (Str)), the value in the register must be in the image of the gates.

In [GJ16, Def. 3], another equation called *timelessness* was presented which models another form of retiming. This equation is derivable from the equations seen so far.

Lemma 103 (Timelessness)**.** *For any gate* , *in $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$.*

Proof.



□

We have now defined all the local equations we need. Unfortunately, we do require some 'non-local' equations to show how the trace interacts with circuits. One is the (IF) equation from Section 5.2, while the other is new.

Lemma 104. (DD) *in Fig. 10 holds in $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$.*

Proof. Since the circuit has no outputs, the output of every state is equal. □

This equation is necessary as the trace can be used to make circuits with no outputs without using the generator. The (IF) equation can only unfold the trace when the underlying circuit is combinational, so an equation to eliminate redundant delay-guarded traces is required.

Definition 105. *Let $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$ be defined as*

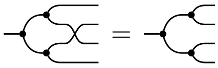
$$\mathbf{SCirc}_{\Sigma}/(\mathcal{C}_{\mathcal{I}} + \mathcal{B} + \mathcal{L} + (\mathsf{IF}) + (\mathsf{DD})).$$

We will conclude this section by showing that $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$ is *Cartesian*. This was shown informally in [GJ16, Thm. 6]; here we restate it in the context of the local equational theory.

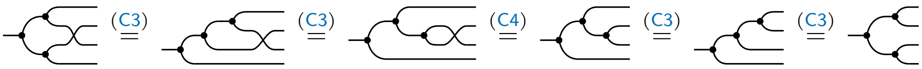
Definition 106 (Cartesian category)**.** *A PROP is* Cartesian *if its tensor product is given by the Cartesian product.*

Effectively this means the fork and stub are *natural*: they satisfy the equations in Fig. 9. To show this is the case for $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$ we first need a lemma.

Lemma 107. *The following is a valid equation in* $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$*:*



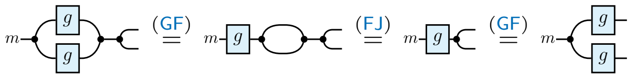
Proof.



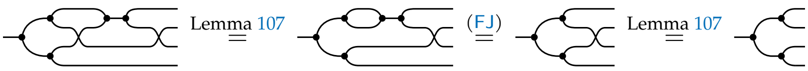
∎

Lemma 108. *For any sequential circuit* m–F–$^{n}_{p}$*, the* (JF) *equation in Fig.* 11 *is valid in* $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$*.*

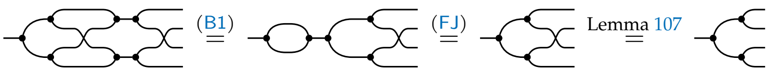
Proof. This is by induction over the structure of the circuit F. First we must check the base cases. For g:



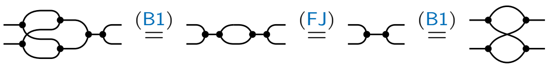
For the fork we first check if only one of the outputs are joined:



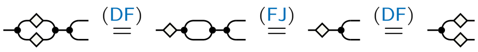
And now if both outputs are joined:



For the join:



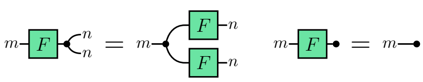
For the delay:



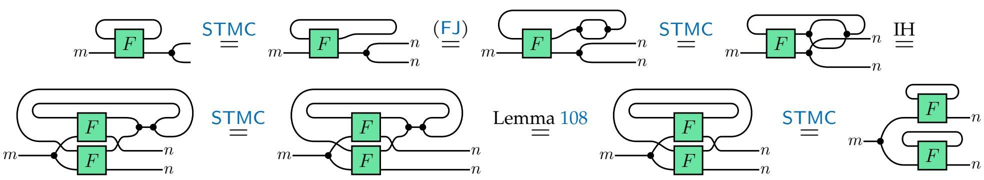
The proofs for the stub, the identity and the swap are trivial, as are the inductive cases for composition and tensor. For the inductive case for the trace, axioms of STMCs are applied in reverse to create two 'global traces', and then the inductive hypothesis is applied to reach the final result. ∎

Theorem 109. $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$ *is Cartesian.*

Proof. To show this we need to show that the two naturality equations hold in $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$:



The naturality of the copy for the generators is immediate by (GF) for gates; by identity for the fork; by (B3) for the join; by (C1) for the stub; by (F) for values; and (DF) for delays. It is also immediate for composition and tensor. For trace it is more involved:



The naturality of the stub for the generators is immediate by (GS) (for the gates), (C1) (for the fork), (B3) (for the join), (S) (for the values) and (DD) (for the delay). It is also trivial for composition and tensor. For the trace the circuit can be brought into Mealy form, followed by using (IF) and (DD) to reduce the circuit to a stub. ∎

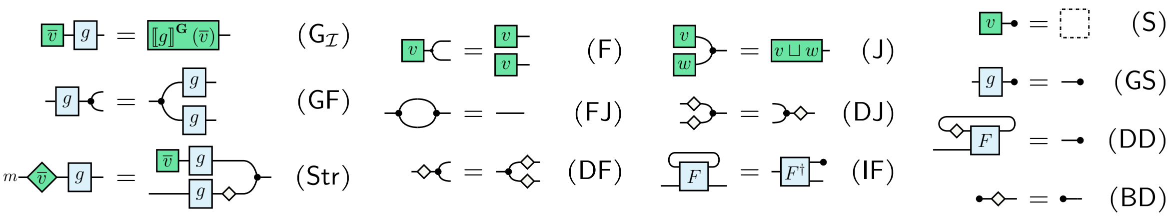
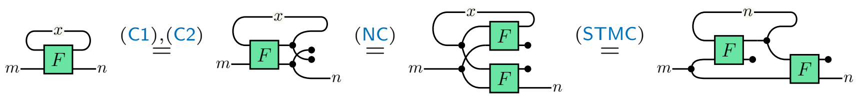


Figure 10: Axioms of $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$, alongside those in Fig. 8.

A category that is Cartesian and traced is known as a *dataflow* category [CŞ90]. These categories are interesting because any dataflow category admits a fixpoint operator [Has97, Thm. 3.1]. Subsequently, there is an important equation that can be derived in any dataflow category.

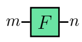
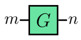
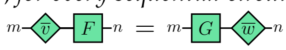
Lemma 110 (Unfolding [Has97; Has99][1]). *The* (UF) *equation in Fig. 11 is valid in* $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$.

Proof. 

□

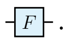
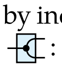
6.2 Productivity

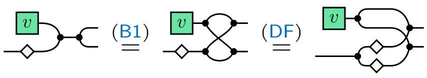
When developing a circuit it is (usually) tested to verify that its behaviour is correct. One way to do this is to provide some inputs and observe the stream of outputs.

Definition 111 (Productivity). *A quotient of* $\mathbf{SCirc}_{\Sigma}$ *is* productive *if, for every sequential circuit*  *and values* $\overline{v}$, *there exists a sequential circuit*  *and values* $\overline{w}$ *such that* .

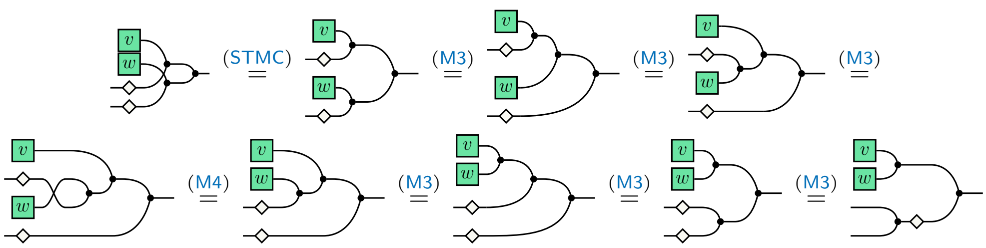
We will now show that $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$ is productive, by deriving an equation using the axioms of $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$, that shows how a sequential circuit processes a value.

Lemma 112 (Generalised streaming). *The* (GStr) *equation in Fig. 11 is valid in* $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$.

Proof. This is by induction on the structure of . First the base cases. The case for the gate is immediate by (Str). For :



For :



The case for  is trivial, and the case for  follows by (C1) and (BD). The cases for  and  follow by axioms of STMCs. Since the underlying circuit is combinational, for the inductive cases we just need to check composition and tensor, which are also trivial. □

Theorem 113 (Cycle equation). *The* (Cycle) *equation in Fig. 11 is valid in* $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$.

[1] Hasegawa also attributes this result to Hyland.

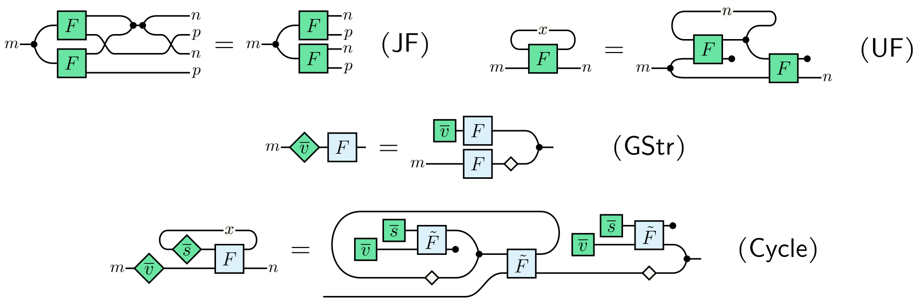
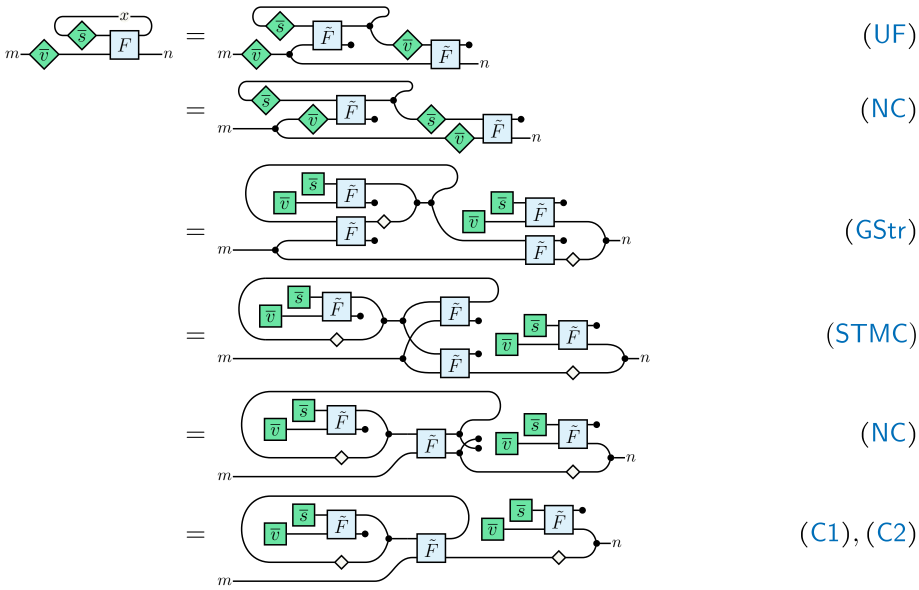
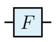


Figure 11: Equations valid in $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$.

Proof.

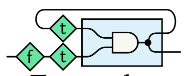


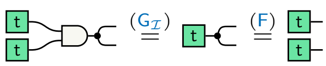
□

The underlying coalgebraic structure of the circuit can be seen in the right of (Cycle). Of the three copies of , the leftmost and the rightmost represent a transition and an output given the current state and input. The copy in the middle is the Mealy function applied to the new state and input.

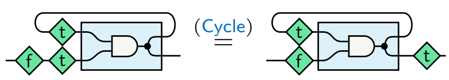
Theorem 114. $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$ *is productive.*

Proof. Since we are only concerned with delay-guarded circuits, the circuit can be put into Mealy form. Then the cycle equation (Cycle) can be applied. Finally, the equations in $\mathcal{C}_{\mathcal{I}}$ can be used to reduce the transition and output copies of the combinational core to values. □

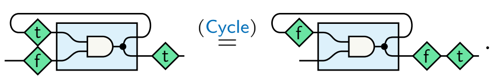
Example 115. Consider the following circuit in Mealy form: . We will apply (Cycle) twice in order to find the first two elements of the output stream. To avoid repetition, we will compute the transition and output concurrently. For the first input element:



So the both the next state and the output are t:



Repeating the procedure again gives us



To conclude this section, we will confirm that this method of reasoning is analogous to taking outputs and stream derivatives in the stream semantics.

Lemma 116. *For any sequential circuit* ${}^{m}\!-\boxed{F}-^{n}$ *and values* $\overline{v} \in \mathbf{V}^m$, *then* $\llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\overline{v} :: \sigma) = \llbracket -\langle\overline{v}\rangle-\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)$.

Proof. By Notation 39 and Definition 24,

$$\llbracket -\langle\overline{v}\rangle-\boxed{F}- (\sigma)(i) \rrbracket^{\mathbf{S}}_{\mathcal{I}} = \begin{cases} \llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\overline{v}) & \text{if } i = 0 \\ \llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma(k)) & \text{if } i = k+1 \end{cases}$$

so clearly $\llbracket -\langle\overline{v}\rangle-\boxed{F}- (\sigma)(i) \rrbracket^{\mathbf{S}}_{\mathcal{I}} = \llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\overline{v} :: \sigma)$. □

Lemma 117. *For any sequential circuit* ${}^{m}\!-\boxed{F}-^{n}$ *and values* $\overline{w} \in \mathbf{V}^n$, *then* $\overline{w} :: \left(\llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)\right) = \left\llbracket -\boxed{F}-\langle\overline{w}\rangle- \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)$.

Proof. By Notation 39 and Definition 24,

$$\left\llbracket -\boxed{F}-\langle\overline{w}\rangle- (\sigma)(i) \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} = \begin{cases} \overline{w} & \text{if } i = 0 \\ \llbracket -\boxed{G}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma(k)) & \text{if } i = k+1 \end{cases}$$

so clearly $\left\llbracket -\boxed{F}-\langle\overline{w}\rangle- (\sigma) \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} = \overline{w} :: \left(\llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)\right)$. □

Theorem 118. *Given a delay-guarded sequential circuit* ${}^{m}\!-\boxed{F}-^{n}$ *and values* $\overline{v}$, *if* $-\langle\overline{v}\rangle-\boxed{F}- \;=\; -\boxed{G}-\langle\overline{w}\rangle-$ *in* $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$, $\left(\llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}\right)[\overline{v}] = \overline{w}$ *and* $\left\llbracket \left(\llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}\right)_{\overline{v}} \right\rrbracket^{\mathbf{S}}_{\mathcal{I}} = \llbracket -\boxed{G}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}$.

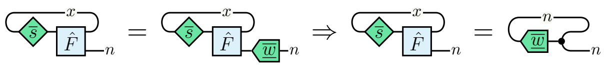
Proof. Assume that ${}^{m}\!-\langle\overline{v}\rangle-\boxed{F}-^{n} \;=\; {}^{m}\!-\boxed{G}-\langle\overline{w}\rangle-^{n}$. First we show that $\left(\llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}\right)[\overline{v}] = \overline{w}$.

$$\begin{aligned}
(\llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}})[\overline{v}] &= \mathsf{i}\left(\llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\overline{v} :: \sigma)\right) && \text{Definition 16} \\
&= \llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\overline{v} :: \sigma)(0) && \text{Definition 14} \\
&= \llbracket -\langle\overline{v}\rangle-\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)(0) && \text{Lemma 116} \\
&= \left\llbracket -\boxed{G}-\langle\overline{w}\rangle- \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)(0) && \text{Assumption} \\
&= \left(\overline{w} :: \llbracket -\boxed{G}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)\right)(0) && \text{Lemma 117} \\
&= \overline{w}
\end{aligned}$$

For the stream derivative:

$$\begin{aligned}
(\llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}})_{\overline{v}} &= \sigma \mapsto \mathsf{d}\left(\llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\overline{v} :: \sigma)\right) && \text{Definition 16} \\
&= \sigma \mapsto \left(i \mapsto \llbracket -\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\overline{v} :: \sigma)(i+1)\right) && \text{Definition 14} \\
&= \sigma \mapsto i \mapsto \llbracket -\langle\overline{v}\rangle-\boxed{F}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)(i+1) && \text{Lemma 116} \\
&= \sigma \mapsto i \mapsto \left\llbracket -\boxed{G}-\langle\overline{w}\rangle- \right\rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)(i+1) && \text{Assumption} \\
&= \sigma \mapsto i \mapsto \left(\overline{w} :: \left(\llbracket -\boxed{G}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)\right)\right)(i+1) && \text{Lemma 117} \\
&= \sigma \mapsto i \mapsto \mathsf{d}\left(\overline{w} :: \left(\llbracket -\boxed{G}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)\right)\right)(i) && \text{Definition 14} \\
&= \sigma \mapsto i \mapsto \llbracket -\boxed{G}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}(\sigma)(i) && \text{Definition 14} \\
&= \llbracket -\boxed{G}- \rrbracket^{\mathbf{S}}_{\mathcal{I}}
\end{aligned}$$

□



(CFix)

Figure 12: The closed fixpoint equation.

6.3 Full reduction

Using the (IF) equation, non-delay-guarded feedback can be exhaustively unfolded. This is not the case for delay-guarded feedback, because a circuit may never 'settle' on one internal configuration, but rather alternate between multiple states.

Recall that the semantics of circuits are that of stream functions with finitely many stream derivatives. In the case where there are no inputs, this becomes a stream with a finite prefix followed by a periodic segment: this means we can specify a closed circuit purely by two finite sequences.

We call such a finite sequence a *waveform*. Finite-length sequences are written with underline, e.g. $\underline{v} := \mathsf{t} :: \mathsf{f} \in \mathbf{V}^\star$. This will also be combined with the overline notation for sequences of powers of values in $(\mathbf{V}^m)^\star$, e.g. $\underline{\overline{v}} := (\mathsf{t}, \mathsf{f}) :: (\mathsf{f}, \mathsf{t}) \in (\mathbf{V}^2)^\star$.

Notation 119 (Waveform)**.** *The empty waveform is defined as* $:=$ *. Given values* $\overline{v} \in \mathbf{V}^n$ *and sequence* $\underline{\overline{w}} \in (\mathbf{V}^n)^\star$*, the waveform for sequence* $\overline{v} :: \underline{\overline{w}}$ *is drawn as* $:=$ *.*

Theorem 120 (Closed fixpoint equation)**.** *For combinational circuit and* $\overline{v} \in \mathbf{V}^n$*, the* (CFix) *equation in Fig. 12 holds in* $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$*.*

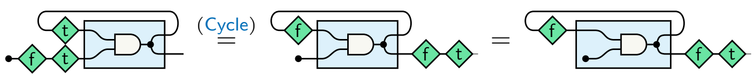
Proof. If $=$ then $[\![\ ]\!]^{\mathbf{S}}_{\mathcal{I}} = [\![\ ]\!]^{\mathbf{S}}_{\mathcal{I}}$. Therefore $[\![\ ]\!]^{\mathbf{S}}_{\mathcal{I}} =$ $\underline{\overline{w}} :: \underline{\overline{w}} :: \underline{\overline{w}} :: \cdots$, which is clearly also the semantics of . Since these have the same semantics, they are also equal by (Bisim) by Theorem 97. □

Definition 121. *Let* $\mathbf{SCirc}^{\mathcal{R}}_{\Sigma,\mathcal{I}} := \mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}/$(CFix).

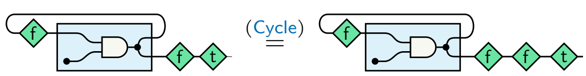
Corollary 122. *For any closed sequential circuit , there exist sequences* $\underline{\overline{v}}, \underline{\overline{w}}$ *such that* $=$ *is valid in* $\mathbf{SCirc}^{\mathcal{R}}_{\Sigma,\mathcal{I}}$*.*

With the addition of this equation, it is possible to reduce any closed circuit to its output stream of values. However, there is one nuance to consider: when evaluating the inputs of a circuit, the repeating segment may have already begun.

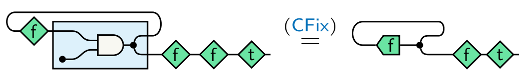
Example 123. Recall the circuit from Example 115, in which we performed two cycles. In order to fully reduce this circuit, we will now cap off the input. Since the generator is in $\mathbf{CCirc}_\Sigma$, it can be brought inside the combinational core.



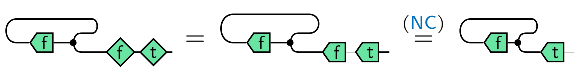
Performing (Cycle) brings us to:



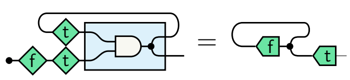
The state f has already reoccurred, so the complete behaviour of the circuit has been specified. We can now apply (CFix) to reduce the periodic segment:



We now have a circuit constructed solely of waveforms but if one looks closely this is *not* the minimal prefix. This can be minimised by using (NC).



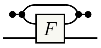
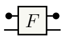
So in total we have shown that:



7 Conclusion, related and further work

Algebraic (categorical) and diagrammatic semantics for combinational Boolean circuits (i.e. no feedback and no delays) 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.

With the semantics rigorous, we have defined some categories suitable for equational reasoning. The first, $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$, contains the (Bisim) equation so that any two circuits with the same semantics are equal. The second, $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$, is constructed primarily from local equations which makes it more suitable for an operational semantics. By quotienting $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$ by one more equation to yield $\mathbf{SCirc}^{\mathcal{R}}_{\Sigma,\mathcal{I}}$, it can be shown that any closed circuit is equal to a circuit specified by two finite sequences of values.

String diagrams as a graphical syntax for monoidal categories were 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; BP22], dynamical systems [BE15; FSR16], electrical circuits [BS22] and automatic differentiation [Alv+21]. 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*, which effectively makes the wires bidirectional. This means that the trace is constructed as , which degenerates to  in our *Cartesian* setting.

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 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 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 this paper, 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 handle fixpoint, but also assemble (monotone) Mealy machines into a PROP in order to use them as a conduit between stream functions and sequential circuits.

Throughout this paper, we have often made use of 'syntactic sugar' to make diagrams clearer, such as combining multiple wires or boxes into one. This could be made formal using the *strictifiers* of [WGZ22]. Another formal syntactic aid could be the use of *layered explanations* [LZ22]: this would allow for viewing circuits at different levels of abstraction.

Reasoning with string diagrams is not an efficient syntax to work with computationally. For an efficient operational semantics, string diagrams must be translated into combinatorial graphs: this was touched on informally in [GJL17a], which used *framed point graphs* [Kis12]. Recent work in string diagram rewriting [Bon+22a; Bon+22b; Bon+22c] has used *hypergraphs* to perform rewriting modulo Frobenius structure. This framework has been adapted for traced categories [Kay21] and categories with a (co)monoid structure [FL22; MZ22]. Since these are the two main structures at play in our setting, the two frameworks could be combined for rewriting sequential circuits. Work on *hierarchical* hypergraphs [Alv+22] is also of interest and could be a way of formally handling the subcircuits involved when using the equations of $\mathbf{SCirc}^{\mathcal{F}}_{\Sigma,\mathcal{I}}$ and $\mathbf{SCirc}^{\mathcal{L}}_{\Sigma,\mathcal{I}}$.

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.* Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004. July 2004, pp. 415–425. DOI: 10.1109/LICS.2004.1319636.

[Alv+21] Mario Alvarez-Picallo et al. *Functorial String Diagrams for Reverse-Mode Automatic Differentiation*. July 28, 2021. DOI: 10.48550/arXiv.2107.13433. arXiv: 2107.13433 [cs].

[Alv+22] Mario Alvarez-Picallo et al. "Rewriting for Monoidal Closed Categories". In: *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*. Ed. by Amy P. Felty. Vol. 228. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 29:1–29:20. ISBN: 978-3-95977-233-4. DOI: 10.4230/LIPIcs.FSCD.2022.29.

[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)*. 2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). June 2019, pp. 1–12. DOI: 10.1109/LICS.2019.8785877.

[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 II: Rewriting with Symmetric Monoidal Structure". In: *Mathematical Structures in Computer Science* 32.4 (Apr. 2022), pp. 511–541. ISSN: 0960-1295, 1469-8072. DOI: 10.1017/S0960129522000317.

[Bon+22c] 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.

[BP22] Guillaume Boisseau and Robin Piedeleu. "Graphical Piecewise-Linear Algebra". In: *Foundations of Software Science and Computation Structures: 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, Apr. 4, 2022, pp. 101–119. ISBN: 978-3-030-99252-1. DOI: 10.1007/978-3-030-99253-8_6.

[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.

[BS22] Guillaume Boisseau and Paweł Sobociński. "String Diagrammatic Electrical Circuit Theory". In: *Electronic Proceedings in Theoretical Computer Science* 372 (Nov. 3, 2022), pp. 178–191. ISSN: 2075-2180. DOI: 10.4204/EPTCS.372.13. arXiv: 2106.07763 [cs].

[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: Capturing Rank Width*. Oct. 3, 2022. DOI: 10.48550/arXiv.2205.08916. arXiv: 2205.08916 [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. arXiv: 2204.02284 [cs, math, stat].

[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.

[Gin14] Abraham Ginzburg. *Algebraic Theory of Automata*. Academic Press, June 25, 2014. 176 pp. ISBN: 978-1-4832-2516-6. Google Books: MCijBQAAQBAJ.

[GJ16] Dan R. Ghica and Achim Jung. "Categorical Semantics of Digital Circuits". In: *2016 Formal Methods in Computer-Aided Design (FMCAD)*. 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.

[Has99] Masahito Hasegawa. *Models of Sharing Graphs*. London: Springer, 1999. ISBN: 978-1-4471-1221-1 978-1-4471-0865-8. DOI: 10.1007/978-1-4471-0865-8.

[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.

[LZ22] Leo Lobski and Fabio Zanasi. *String Diagrams for Layered Explanations*. July 8, 2022. DOI: 10.48550/arXiv.2207.03929. arXiv: 2207.03929 [cs, math].

[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.

[MZ22] Aleksandar Milosavljevic and Fabio Zanasi. *String Diagram Rewriting Modulo Commutative Monoid Structure*. Apr. 8, 2022. DOI: 10.48550/arXiv.2204.04274. arXiv: 2204.04274 [cs, math].

[Rie04] Marc D. Riedel. "Cyclic Combinational Circuits". PhD thesis. United States – California: 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)*. 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)*. 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.

[WGZ22] Paul Wilson, Dan Ghica, and Fabio Zanasi. *String Diagrams for Non-Strict Monoidal Categories*. Jan. 29, 2022. DOI: 10.48550/arXiv.2201.11738. arXiv: 2201.11738 [math].

[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.