Normalisation by evaluation for digital circuits
Extended abstract

George Kaye, Dan R. Ghica and David Sprunger

November 17, 2021

Motivation. Digital circuits are ubiquitous in today's society, so it is essential that we have easy ways to verify their correctness and reason with them. Conventionally, this is done by translation into an executable model such as an automaton, which can be simulated to observe its behaviour. Conversely, reasoning in software is often performed *syntactically*, where programs are formulated equationally and can be reduced step by step by identifying *redexes* in terms. These redexes are replaced with simpler terms in order to reach some normal form. This is known as *operational semantics* [8].

In [4], an operational semantics for digital circuits was presented, in which circuits with delay and feedback are modelled as morphisms in a free traced cartesian (dataflow) category [3, 2]. One benefit of this operational framework is that it allows reasoning about circuits with *combinational feedback* [4, Example 1], in which a circuit has a non-delay-guarded feedback loop but the resulting circuit is actually combinational. In many circuit design tools, feedback must always be guarded by a delay and these circuits cannot be modelled.

However, this does raise a problem when we have circuits with instant feedback that is *not* combinational. In the original semantics, such circuits would never produce a value, but can only be unfolded infinitely. In this extended abstract, we look at the 'missing link' from our categorical framework that allows us to deal with this problem. Moreover, we show how adding it to our semantics gives us soundness and completeness for *periodic streams*, the denotational semantics for closed circuits. This verifies our intuition regarding the correctness of our categorical framework, and also presents a strategy of *normalisation by evaluation* for our digital circuits.

Combinational circuits. As in [4], we model digital circuits as freely generated morphisms in a traced cartesian (dataflow) prop: a category with objects the natural numbers and tensor product as addition.

Definition 1. *For a finite lattice* $\mathbf{V}$*, let* $\mathbf{Circ_V}$ *be a categorical signature with objects the natural numbers and a finite set of morphisms from three classes:* ***values*** $v : 0 \to 1$ *for each* $v \in \mathbf{V}$*;* ***gates*** $g : m \to 1$*; and* ***structures*** *for forking* $\prec : 1 \to 2$*, joining* $\succ : 2 \to 1$ *and stubbing* $\wr : 1 \to 0$ *wires.*

Traditionally, we use a four-value lattice $\mathbf{V}$ containing t for high and f for low in addition to $\bot$ and $\top$, such that $\mathsf{t} \sqcup \mathsf{f} = \top$ and $\mathsf{t} \sqcap \mathsf{f} = \bot$. We can define forks, joins and stubs for buses of arbitrary widths by combining them with identities and symmetries: we denote these as Δ, ∇ and $\diamond$ respectively. We write $\mathbf{v}$ for a tensor of values $v_0, v_1, \cdots, v_{n-1}$. In our string diagrams, we draw $\prec$, $\succ$ and $\wr$, as well as their arbitrary width versions, as black dots $\bullet$. The actual morphism should be clear from context.

Definition 2 (Combinational circuits)**.** *Let* $\mathbf{CCirc}_V$ *be the free symmetric monoidal category over* $\mathbf{Circ}_V$ *and monoidal signature* $(\mathbb{N}, +, 0)$ *subject to the equations in Figure 1, quotiented such that circuits with the same input-output behaviour are equal.*

The first three equations cover copying a value, joining two values by coalescing them, and stubbing a wire. The last equation states that gates are *extensional*: their outputs are solely determined by their inputs. Using these equations we can derive an important lemma:

Lemma 3 (Extensionality)**.** *For a circuit* $f : 0 \to n \in \mathbf{CCirc}_V$*, there exists a tensor* $\boldsymbol{v}$ *such that* $f = \boldsymbol{v}$*.*

Quotienting by input-output behaviour provides some interesting structure to our category. For example, Δ and $\diamond$ are the copy and discard maps of a cartesian category. Moreover $(\succ, \bot, \prec, \wr)$ forms a *bialgebra*.

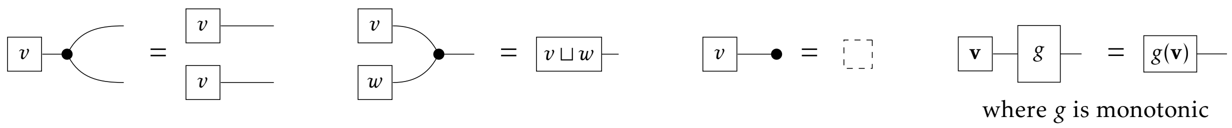


Figure 1: The axioms of combinational circuits, where v and w are values.

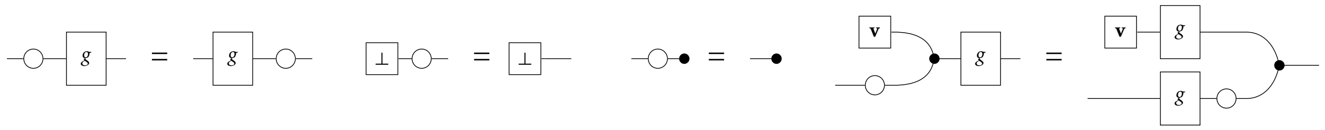


Figure 2: The axioms of sequential circuits, where g is a gate.

Sequential circuits. Combinational circuits are not very interesting. We now turn our attention to *sequential circuits* with delay and feedback. We model delay as a structural morphism $\delta_n : 1 \to 1$, parameterised by $n \in \mathbb{N}$: the duration of the delay. We draw it as a white circle in our string diagrams. Adding feedback is easy: we simply add a trace to the category.

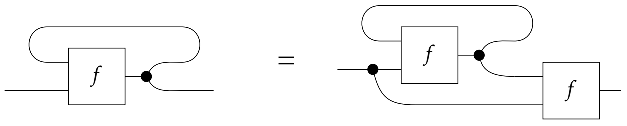
Definition 4 (Seqential circuits)**.** *Let* ***SCirc***$_V$ *be the category obtained by freely extending* **CCirc**$_V$ *with a trace operator and a new generator* $\delta_n : 1 \to 1$ *subject to the equations in Figure 2.*

The first three delay axioms should be easy to interpret. The fourth (*Streaming*) is more subtle: it means that to process a *waveform* with an instantaneous value at its 'head' and some other delayed component, we copy the gate. One copy is for whatever is happening 'now', and the other is for what is happening 'later'. We can then join the outputs of the two copies.

SCirc$_V$ is obviously a traced monoidal category. However, we can go one better:

Theorem 5 ([4])**.** ***SCirc***$_V$ *is a traced cartesian category with* Δ *and* $\diamond$ *as the diagonal and unit.*

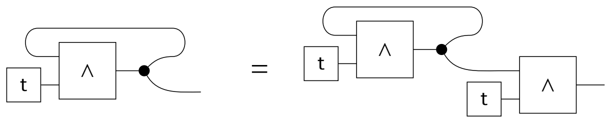
Traced cartesian categories (also known as *dataflow categories* [9, Section 6.4]) arise naturally in computer science through *fixpoints* [3, 2]. In fact, a category is only traced cartesian if it admits a *conway operator* [7]. This allows us to use the *unfolding* rule, which can be derived using the diagonal and the trace.



This powerful rule is a key part of the diagrammatic semantics for digital circuits detailed in [6].

Instant feedback. As we have established in the motivation, using this categorical framework allows us to reason with circuits with *combinational feedback*, in which there *is* an instant feedback loop, but the circuit is actually combinational. However, problems can arise when considering circuits with instant feedback that is *not* combinational. This phenomenon is illustrated in the following example.

Example 6. *Consider the circuit* $\mathsf{t} \mathbin{⨟} Tr^1(\wedge \mathbin{⨟} \prec)$*. By the axioms of circuits specified in [4], the only thing we can do to this circuit is unfold it, which results in*



We could keep unfolding but this would just result in the circuit getting larger. There is never any opportunity for other rewrites, so we are stuck.

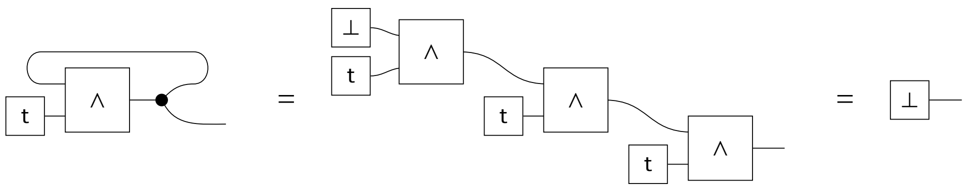


Figure 3: Applying the instant feedback axiom.

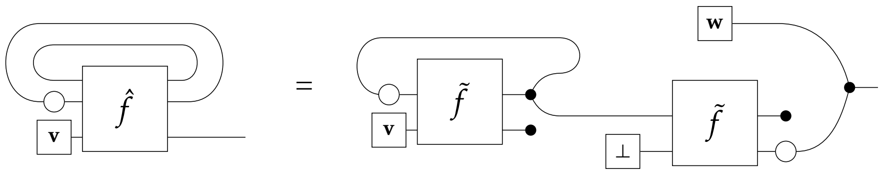


Figure 4: We can always produce an instantaneous value from a circuit thanks to Theorem 7.

How can we solve this problem? We cannot simply enforce that all feedback is delay-guarded, as this would mean we could not model the circuits with combinational feedback.

Fortunately, there is a solution. Due to the Kleene fixed-point theorem [10, p.24], the fact that our values form a finite lattice and that all the gates in our setting are monotone, the circuit will always stabilise under a finite number of iterations. This number is at most one more than the length of the longest chain of the lattice.

Therefore if we have some subcircuit f with an instant feedback loop, we can actually model this by iterating f a finite number of times, using $\perp$ as the initial value of the feedback loop. Figure 3 illustrates this in the case of Example 6 for our usual lattice $\mathbf{V}$, in which the length of the longest chain is 2.

Armed with this axiom, we can refine the original productivity result [5, Theorem 24]. Rather than restricting ourselves to delay-guarded circuits, we can use it for *any* closed circuit.

Theorem 7 (Productivity). *For a closed circuit $f : 0 \to n$, there exists a tensor of values $\boldsymbol{w}$ and morphism $g : 0 \to n$ such that $f = \boldsymbol{w} \otimes g \,⨾\, \nabla_n$.*

This is due to instant feedback, unfolding, and streaming, and is illustrated in Figure 4.

Streams. Using productivity, we are able to reduce a closed circuit to an instantaneous tensor of values and some delayed subcircuit. We call this a *waveform*, and write $\mathcal{W}(f)[i]$ for the tensor of values obtained by applying productivity i times. This gives us a *stream* of values $\mathcal{W}(f)[0], \mathcal{W}(f)[1], \mathcal{W}(f)[2], \cdots$.

Streams were already examined as a concrete semantics for our category of digital circuits in [6, Section 2.4]. However, this was only used as a method to verify the intuition behind the axioms. We want to go one step further, and characterise an *isomorphism* between circuits and streams. For now, we concentrate on closed circuits and *periodic* streams.

Definition 8 (Periodic stream). *A periodic stream is a stream that has a finite prefix followed by an infinitely repeating finite segment, i.e. a stream of the form $(v_0, v_1, \cdots, v_{p-1}, w_0, w_1, \cdots, w_{r-1}, w_0, w_1, \cdots, w_{r-1}, w_0, w_1, \cdots$.*

This follows intuition: our circuits contain only a finite number of delay elements (registers) and the value lattice we operate in is also finite, so at some point the internal state must repeat. For a stream σ, we write $\sigma[i]$ for the ith element, i.e. the head of the ith stream derivative.

To interpret circuits as streams, we must first examine a concrete implementation of our circuits as *Mealy machines*. We can then make the jump to streams, as we shall see later.

Definition 9 (Mealy machines [1]). *For lattices $\boldsymbol{M}, \boldsymbol{R}, \boldsymbol{N}$, a Mealy machine with interface $(\boldsymbol{M}, \boldsymbol{N})$ is a tuple $(\boldsymbol{X}, \langle O, T \rangle)$ where $O : \boldsymbol{R} \to \boldsymbol{N}^{\boldsymbol{M}}$ and $T : \boldsymbol{R} \to \boldsymbol{R}^{\boldsymbol{M}}$ are Scott-continuous functions representing* outputs *and* transitions *respectively. For a Mealy machine A, we can* initialise *it with a given start state $s_0 \in \boldsymbol{R}$. We call the tuple (A, s_0) an* initialised Mealy machine.

Mealy machines can be interpreted as coalgebra of the functor $F : \mathbf{R} \to (\mathbf{N} \times \mathbf{R})^{\mathbf{M}}$ in **Lat**, the category of lattices. This means that there is a standard notion of homomorphism [1] and, importantly, the notion of a *final Mealy machine*. The state space of this Mealy machine is the set of streams of $\mathbf{N}$, and its output and transition functions are the initial value and stream derivative respectively.

This means that there is a unique function that transforms a Mealy machine into its corresponding stream. Moreover, any two Mealy machines with the same corresponding stream must therefore be *bisimilar*: observationally equivalent. Obtaining a stream in this way from a Mealy machine provides us with the following crucial result:

Proposition 10. *A closed Mealy machine generates a periodic stream if* $\mathbf{R}$ *is finite.*

Interpretation. In our context we are only concerned with Mealy machines where $\mathbf{M}$, $\mathbf{R}$ and $\mathbf{N}$ are all sets of tuples over the same lattice $\mathbf{V}$. We write a machine with inputs $\mathbf{V}^m$, states $\mathbf{V}^r$ and outputs $\mathbf{V}^n$ as $m \xrightarrow{r} n$. Notably, as $\mathbf{V}$ is finite, so is $\mathbf{V}^r$.

To establish a link between our circuits and streams, we must interpret them as Mealy machines. Therefore we must define the operations of composition, tensor and trace. Composition and tensor are fairly obvious; for trace, we use a fixpoint operator to 'simulate' the circuit over a finite number of iterations, similarly to how we defined the instant feedback axiom.

Definition 11. *Let* $\mathbf{Mealy}_V$ *be the prop with morphisms* $m \to n$ *the initialised Mealy machines over* $\mathbf{V}$*. Let* $\mathbf{Stream}_\mathbf{V}$ *be the prop with morphisms* $m \to n$ *the streams of functions* $\mathbf{V}^m \to \mathbf{V}^n$ *and let* $\mathbf{CStream}_\mathbf{V}$ *be the subcategory containing only the morphisms* $0 \to n$*, i.e. streams of* $\mathbf{V}^n$*. Finally, let* $\mathbf{PCStream}_\mathbf{V}$ *be the subcategory of* $\mathbf{CStream}_\mathbf{V}$ *containing only the periodic streams.*

Proposition 12. $\mathbf{Mealy}_V$ *is a traced cartesian category.*

Definition 13. *Let* $[\![-]\!] : \mathbf{SCirc}_V \to \mathbf{Mealy}_V$ *be the identity-on-objects traced monoidal functor that translates circuits into the corresponding machine in* $\mathbf{Mealy}_V$*. Then let* $\nu(-) : \mathbf{Mealy}_V \to \mathbf{Stream}_\mathbf{V}$ *be the identity-on-objects traced monoidal functor that produces the unique stream corresponding to a Mealy machine.*

The transition is defined inductively over the structure of the term: while we omit the details, we will sketch it. The set of states are the possible contents of the value and delay generators. For example, any value generator will initially contain a value, but will permanently transition to $\bot$ after 'releasing' it; conversely, any delay will initially contain $\bot$, but may store a value in subsequent ticks. Gates and structural morphisms are stateless machines that simply output the function corresponding to its behaviour. Composition, tensor and trace use the operations from $\mathbf{Mealy}_V$. Since all the axioms of $\mathbf{SCirc}_V$ hold in $\mathbf{Mealy}_V$, we can conclude the following:

Theorem 14. *For any* $f, g \in \mathbf{SCirc}_V$*, if* $f = g$ *then* $\nu([\![f]\!]) = \nu([\![g]\!])$*.*

Isomorphism. We can translate from circuits to streams: can we go in the opposite direction? For the case of periodic streams of values, this is quite simple.

Definition 15. *Let* $\langle - \rangle : \mathbf{PCStream}_\mathbf{V} \to \mathbf{SCirc}_V$ *be an identity-on-objects traced monoidal functor, defined for a periodic stream* $(v_0, v_1, \cdots, v_{p-1}, w_0, w_1, \cdots, w_{r-1}, w_0, w_1, \cdots$ *as illustrated in Figure 5.*

This is a waveform! As we mentioned earlier, we can transform any circuit into a waveform, albeit with an arbitrary circuit g as its tail rather than the complete set of explicit values we computed in the stream. We simply need to assert that the values at each tick are the same.

Proposition 16 (Stream-waveform correspondence)**.** *For a circuit* $f : 0 \to n$*,* $\mathcal{W}(f)[i] = \nu([\![f]\!])[i]$*.*

With this key principle we can conclude our final result.

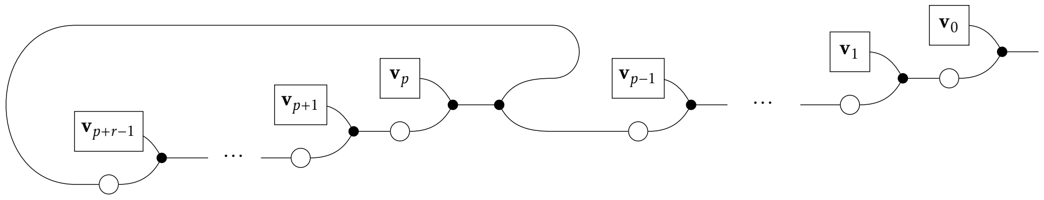


Figure 5: The circuit morphism obtained from a periodic stream.

Theorem 17. *$\mathbf{CSCirc}_V \cong \mathbf{PCStream}_V$.*

The circuit we obtain from the stream interpretation may be in a much simpler form than the original morphism. We can therefore see that translating a closed circuit into a stream and back again is a form of *evaluation by normalisation*. This offers an alternative to the reduction-based framework, which may be more efficient in some cases.

Future work. We have shown that there is indeed a link between closed circuits and periodic streams, and that we can translate between them freely without losing information. One may wonder how the link between circuits and streams is affected if we use circuits that are *not* closed. In this case, the denotational semantics are *stream functions* that take a stream of inputs and return a stream of outputs, rather than just streams of values. Searching for similar results for non-closed circuits remains as future work.

References

[1] Marcello M Bonsangue, Jan Rutten, and Alexandra Silva. "Coalgebraic logic and synthesis of Mealy machines". In: *International Conference on Foundations of Software Science and Computational Structures*. Springer. 2008, pp. 231–245. DOI: 10.1007/978-3-540-78499-9_17.

[2] V. E. Cazanescu and Gheorghe Stefanescu. "Feedback, Iteration, and Repetition". In: *Mathematical Aspects of Natural and Formal Languages*. 1994. DOI: 10.1142/9789814447133_0003.

[3] V. E. Cazanescu and Gheorghe Stefanescu. "Towards a new algebraic foundation of flowchart scheme theory". In: *Fundamenta Informaticae* 13 (1990), pp. 171–210. DOI: 10.5555/97367.97373.

[4] Dan R Ghica and Achim Jung. "Categorical semantics of digital circuits". In: *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design*. FMCAD Inc. 2016, pp. 41–48. DOI: 10.1109/FMCAD.2016.7886659.

[5] Dan R. Ghica, Achim Jung, and Aliaume Lopez. "Diagrammatic Semantics for Digital Circuits". In: *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*. Vol. 82. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 24:1–24:16. DOI: 10.4230/LIPIcs.CSL.2017.24.

[6] Dan R. Ghica, Achim Jung, and Aliaume Lopez. *Diagrammatic Semantics for Digital Circuits (Technical Report)*. 2017. arXiv: 1703.10247 [cs.PL].

[7] Masahito Hasegawa. "Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi". In: *International Conference on Typed Lambda Calculi and Applications*. Springer. 1997, pp. 196–213. DOI: 10.1007/3-540-62688-3_37.

[8] Gordon D. Plotkin. "A structural approach to operational semantics". In: *J. Log. Algebr. Program.* 60-61 (2004), pp. 17–139. DOI: 10.1016/j.jlap.2004.03.002.

[9] Peter Selinger. "A survey of graphical languages for monoidal categories". In: *New structures for physics*. Springer, 2010, pp. 289–355. DOI: 10.1007/978-3-642-12821-9_4.

[10] Viggo Stoltenberg-Hansen et al. *Mathematical theory of domains*. 22. Cambridge University Press, 1994.