Fully abstract categorical semantics for digital circuits
Extended abstract

George Kaye, David Sprunger and Dan R. Ghica

July 20, 2022

Contribution. It is essential that we have ways to verify the correctness of digital circuits and reason with them. Conventionally, this is done by translation into an executable model which can be simulated to observe its behaviour. An alternative approach, used in software, is to reason *syntactically*: programs are formulated equationally and can be reduced step by step. When provided with inputs, the goal of such a system is to apply reductions and derive an output value.

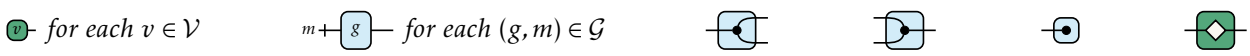
Such an equational system was first presented in [GJ16; GJL17], in which digital circuits with delay and (instant) feedback are modelled as morphisms in a freely generated traced cartesian category, or *dataflow category* [CŞ94; Has97]. However, the presentation was informal and, crucially, not complete: not all circuits could be reduced to a stream of values. Our work brings this project to its conclusion, formalising the categorical semantics and completing the set of equations.

Syntax. Circuits are defined over a *signature*.

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

The distinct elements $\bullet$ and $\circ$ represent a *disconnected wire* (a *lack* of information) and a *short circuit* (*inconsistent* information) respectively. Digital circuits are constructed as morphisms in a freely generated symmetric traced monoidal category (STMC). To aid in the presentation, we shall use the graphical calculus of *string diagrams* [JS91; JSV96; Sel11].

Definition 2 (Sequential circuits)**.** *For a signature Σ, let $\mathbf{SCirc}_\Sigma$ be the symmetric traced monoidal category freely generated over:*



for each $v \in \mathcal{V}$ *for each $(g, m) \in \mathcal{G}$*

In turn each generator represents: *values* that flow through wires; *gates*; constructs for forking, joining and stubbing wires; and finally a generator for *delay*. Circuits obtained by composing generators are drawn as dark green squares 

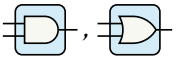
; circuits containing only gates and structural generators are drawn in a lighter blue square 

. To avoid clutter, we occasionally omit the backgrounds of concrete generators in large diagrams.

Semantics. Circuits specified syntactically have no computational content. To add *semantics* to circuits, first the signature must be interpreted in some domain.

Definition 3 (Interpretation)**.** *An interpretation of $\Sigma = (\mathcal{V}, \bullet, \circ, \mathcal{G})$ is a tuple $\mathcal{I} = (\mathbf{V}, \mathcal{I}_\mathcal{V}, \mathcal{I}_\mathcal{G})$ where $(\mathbf{V}, \sqsubseteq, \bot, \top)$ is a finite lattice, $\mathcal{I}_\mathcal{V}$ is a function $\mathcal{V} \setminus \{\bullet, \circ\} \to \mathbf{V} \setminus \{\bot, \top\}$, and $\mathcal{I}_\mathcal{G}$ is a map from each $(g, m) \in \mathcal{G}$ to a monotone function $\overline{g} \colon \mathbf{V}^m \to \mathbf{V}$ such that $\overline{g}(\bot^m) = \bot$ and $\overline{g}(\mathbf{v})$ is in the image of $\mathcal{I}_\mathcal{V}$ for all $\mathbf{v} \in \mathbf{V}^m$.*

Example 4. *Let $\Sigma_\star = (\{\bullet, \mathsf{t}, \mathsf{f}, \circ\}, \bullet, \circ, \{(AND, 2), (OR, 2), (NOT, 1)\})$ be a signature. In $\mathbf{SCirc}_{\Sigma_\star}$, the values are*  

and 
; the gates are 
and 
. Let $\mathbf{V}_\star$ be the lattice $(\{\bot, 0, 1, \top\}, \sqsubseteq)$, with the join defined as $0 \sqcup 1 = \top$ and the meet defined as $0 \sqcap 1 = \bot$. Let $\{\wedge, \vee, \neg\}$ be the Belnap logic *operators [Bel77]: the truth tables are listed in Fig. 1. Let $\mathcal{I}_\star = (\mathbf{V}_\star, \{\mathsf{f} \mapsto 0, \mathsf{t} \mapsto 1\}, \{AND \mapsto \wedge, OR \mapsto \vee, NOT \mapsto \neg\})$.*

The semantics of circuits is that of *stream functions*, which take as input a stream and output a stream. In particular, we are interested in stream functions of the form $(\mathbf{V}^m)^\omega \to (\mathbf{V}^n)^\omega$.

Example 5. *For a value $v \in \mathcal{V}$, the stream function $\mathsf{val}_v \colon (\mathbf{V}^0)^\omega \to \mathbf{V}^\omega$ is defined as $\mathsf{val}_v(\bullet)(0) := \mathcal{I}_\mathcal{V}(v), \mathsf{val}_v(\bullet)(i+1) := \bot$. For a gate $(g, m) \in \mathcal{G}$, the stream function $\mathsf{gate}_g \colon (\mathbf{V}^m)^\omega \to \mathbf{V}^\omega$ is defined as $\mathsf{gate}_g(\sigma)(i) := \mathcal{I}_\mathcal{G}(g)(\sigma(i))$. Finally, the* shift *stream function $\delta \colon \mathbf{V}^\omega \to \mathbf{V}^\omega$ is defined as $\delta(\sigma)(0) := \bot, \delta(\sigma)(i+1) := \sigma(i)$.*

Definition 6. *For a signature $\Sigma = (\mathcal{V}, \bullet, \circ, \mathcal{G})$ and its interpretation $\mathcal{I} = (\mathbf{V}, \mathcal{I}_\mathcal{V}, \mathcal{I}_\mathcal{G})$, let $\mathbf{Stream}_\mathcal{I}$ be the prop freely generated over val_v for each $v \in \mathcal{V}$, gate_g for each $(g, m) \in \mathcal{G}$, and the shift stream δ. Composition and tensor are by function composition and cartesian product; the symmetry swaps input streams.*

Theorem 7. *$\mathbf{Stream}_\mathcal{I}$ is traced.*

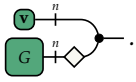
Definition 8. *Let $[\![-]\!]_\mathcal{I} : \mathbf{SCirc}_\Sigma \to \mathbf{Stream}_\mathcal{I}$ be a traced prop morphism, mapping circuits to appropriate stream functions. The details are omitted, see [GKS22].*

If two circuits map to the same semantics in $\mathbf{Stream}_{\mathcal{I}}$, we say they are *extensionally equivalent*, written $F \approx_{\mathcal{I}} G$.

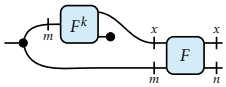
Theorem 9 ([GKS22]). *Let* $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$ *be the category obtained by quotienting* $\mathbf{SCirc}_{\Sigma}$ *by* $\approx_{\mathcal{I}}$. *Then there is an isomorphism of categories* $\mathbf{SCirc}_{\Sigma,\mathcal{I}} \cong \mathbf{Stream}_{\mathcal{I}}$.

Equational reasoning. Circuits of non-equal syntax can have the same semantics as stream functions. However, in general it is prohibitive to check that the corresponding streams for two circuits are equal: it is more efficient to reason *equationally*. Equations are identities that hold in the quotient category $\mathbf{SCirc}_{\Sigma,\mathcal{I}}$: we write $F = G$ if F is equal to G under the equational theory. Note that since we are using string diagrams, the axioms of STMCs are 'absorbed' into the notation and always hold by moving wires and boxes around.

Productivity. A common use of equational reasoning is to take a circuit and reduce it to its stream of output values.

Definition 10 (Productivity). *For a set of equations* $\mathcal{E}$, *a closed sequential circuit* F *is called* productive under $\mathcal{E}$ *if there exist values* $\mathbf{v}^n$ *and sequential circuit* G *such that* $F =_{\mathcal{E}}$  .

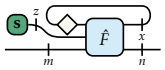
A set of equations was presented in [GJ16]. However, they were not *complete*: these axioms could not necessarily handle circuits with *non-delay-guarded feedback*, in which every feedback loop does not pass through a delay generator. While in some circuits 'instant feedback' is useful [Rie04; MSB12], in other cases it can result in an unproductive circuit. To tackle this, we use the *Kleene fixpoint theorem*: since all the gates in an interpretation are monotone, they have a least fixpoint; since our lattice is finite, we are able to compute it after a finite number of iterations.

Definition 11. *For a combinational circuit* F, *let its* nth iteration F^n *be defined inductively as* $F^0 := \;$  *and* $F^{k+1} :=$  . *Let* $\mathcal{I} = (\mathbf{V}, \mathcal{I}_{\mathcal{V}}, \mathcal{I}_{\mathcal{G}})$ *be an interpretation and let* c *be the length of the longest chain in* $\mathbf{V}$: *the* fixpoint *of* F *in* $\mathcal{I}$, *denoted as* $F^\dagger$, *is defined as* F^{c+1}.

The complete set of equations $\mathcal{C}$ for closed circuits under *any* interpretation is shown in Fig. 2. An important consequence of these is that the *unfolding* rule for circuits with feedback can be derived, illustrated in Fig. 3.

Theorem 12. *Any closed sequential circuit* F *is productive under* $\mathcal{C}$.

By applying productivity, a sequence of values can be obtained for *any* sequential circuit F given some inputs $\mathbf{v}$. This sequence is precisely the corresponding stream obtained using $[\![-]\!]_{\mathcal{I}}$.

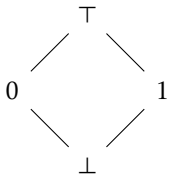
Full abstraction. In the closed case these equations suffice, but when faced with an *open circuit* we must devise another equation. To do this, we view circuits as *state transition machines*: recall that a circuit F is in *global delay form* if it is in the form  . We call the circuit  a *combinational core* of F: this circuit produces its *state transition* and its *output*. To equate circuits with the same behaviour we essentially construct a bisimulation between these state machines.

Proposition 13. *Let* F *and* F *be sequential circuits. Then, if their combinational cores produce the same outputs for all accessible states, then* $F = F$

This equivalence may look obtuse at first glance, but many familiar equations can be derived using it: a selection is shown in Figs. 4 and 5. With this equation, the final result can be shown.

Theorem 14 (Full abstraction). $F =_{\mathcal{C}+\mathsf{MM}} G$ *if and only if* $[\![F]\!]_{\mathcal{I}} = [\![G]\!]_{\mathcal{I}}$.

This allows us to reason *purely equationally* with digital circuits, instead of appealing to the potentially inefficient stream semantics. Even so, this does not immediately yield an *automatic* rewriting framework, as computationally it is difficult to handle the trace. A suitable strategy for tackling this problem was presented in [GJL17] using graph rewriting on *framed point graphs*; a current thread of work is reworking this using recent work on rewriting with *hypergraphs* [Bon+16; Kay21].



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

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

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

Figure 1: The lattice structure on $\mathbf{V}_\star$, and truth tables for the gates in $\Sigma_\star$ under $\mathcal{I}_\star$.

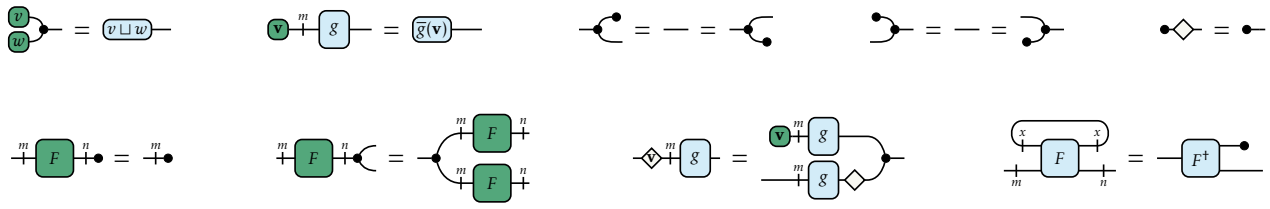


Figure 2: Equations for reducing closed circuits.

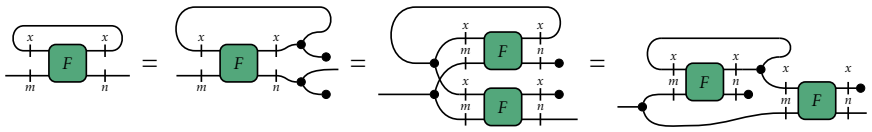


Figure 3: Deriving the *unfolding* rule.

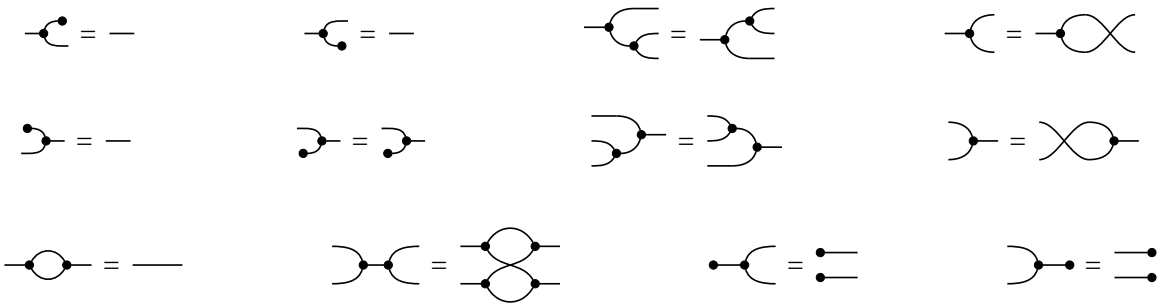


Figure 4: The equations of a *bialgebra*, derivable using the 'Mealy equation'

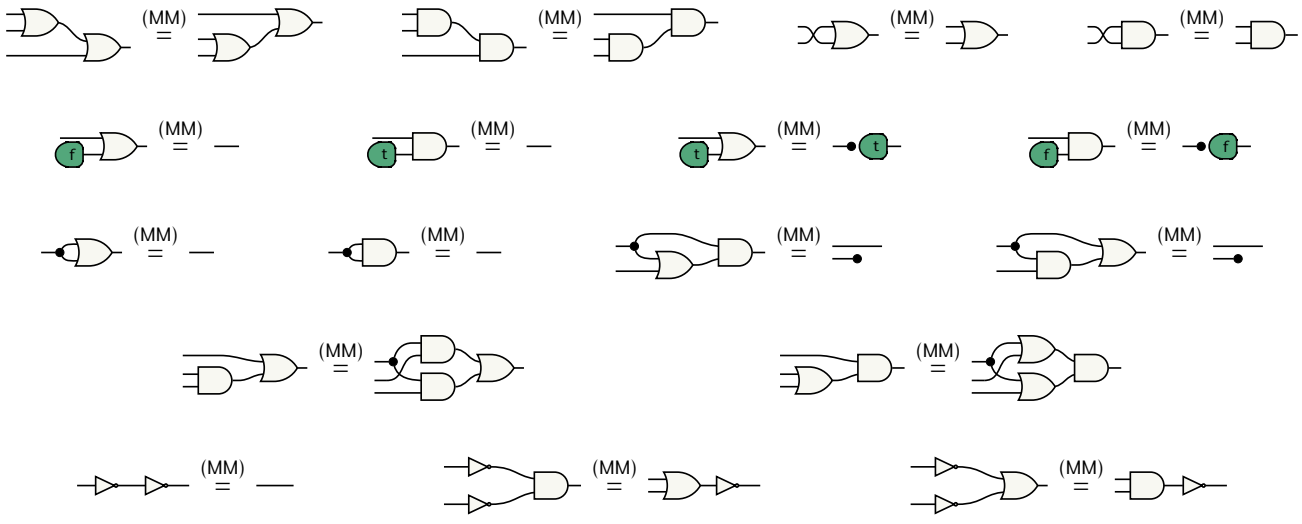


Figure 5: A set of equations derivable using the 'Mealy equation', .

References

[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+16] Filippo Bonchi et al. "Rewriting modulo Symmetric Monoidal Structure". 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. 710–719. ISBN: 978-1-4503-4391-6. DOI: 10.1145/2933575.2935316.

[CŞ94] Virgil Emil Căzănescu and Gheorghe Ştefănescu. "Feedback, Iteration, and Repetition". In: *Mathematical Aspects of Natural and Formal Languages*. Vol. Volume 43. World Scientific Series in Computer Science Volume 43. World Scientific, Oct. 1, 1994, pp. 43–61. ISBN: 978-981-02-1914-7. DOI: 10.1142/9789814447133_0003.

[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.

[GJL17] 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.

[GKS22] Dan R. Ghica, George Kaye, and David Sprunger. "Full Abstraction for Digital Circuits". 2022. arXiv: 2201.10456 [cs, math].

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

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

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

[Kay21] George Kaye. "Rewriting Graphically with Symmetric Traced Monoidal Categories". 2021. arXiv: 2010.06319 [math].

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

[Rie04] Marc D. Riedel. "Cyclic Combinational Circuits". PhD thesis. United States: California Institute of Technology, May 27, 2004. 112 pp. ISBN: 9780496071005. DOI: 10.7907/410B-XR25.

[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.