Diagrammatic Semantics for Digital Circuits*

Dan R. Ghica[1], Achim Jung[2], and Aliaume Lopez[3]

1 **University of Birmingham, United Kingdom**
2 **University of Birmingham, United Kingdom**
3 **ENS Cachan, Université Paris-Saclay, France**

Abstract

We introduce a general diagrammatic theory of digital circuits, based on connections between monoidal categories and graph rewriting. The main achievement of the paper is conceptual, filling a foundational gap in reasoning syntactically and symbolically about a large class of digital circuits (discrete values, discrete delays, feedback). This complements the dominant approach to circuit modelling, which relies on simulation. The main advantage of our symbolic approach is the enabling of automated reasoning about *parametrised circuits*, with a potentially interesting new application to *partial evaluation* of digital circuits. Relative to the recent interest and activity in categorical and diagrammatic methods, our work makes several new contributions. The most important is establishing that categories of digital circuits are Cartesian and admit, in the presence of feedback expressive iteration axioms. The second is producing a general yet simple graph-rewrite framework for reasoning about such categories in which the rewrite rules are computationally efficient, opening the way for practical applications.

1998 ACM Subject Classification B.6.3 Hardware description languages

Keywords and phrases digital circuits, string diagrams, rewriting, operational semantics

Digital Object Identifier 10.4230/LIPIcs.CSL.2017.24

1 Introduction

Of the many differences between the worlds of software and hardware design, a particularly intriguing one is their prevailing modelling methodologies. The workhorse of software reasoning – *operational semantics* [28] – is syntactic and reduction-based. It is essentially an abstract, entirely machine-independent presentation of a programming language which is not required to be faithful to the execution model other than insofar as the final result is concerned. On the other hand, reasoning about hardware relies on having an accurate execution model, akin to what we would call an *abstract machine* in programming languages, usually some kind of automaton [21]. To reason about a circuit, it is translated so that its execution is simulated by the automaton. The abstract machine approach is of course established and useful in programming language theory as well [23], especially in compiler design. But the operational semantics has several advantages over the abstract machine approach, of which perhaps the most important is the ability to evaluate programs which are specified only in part. This is useful because many front-end compiler optimisations are, in one way or another, partial evaluations [7].

Broadly speaking, the main contribution of our paper is to provide an operational semantics for digital circuits, based on diagram rewriting. Our methodology is influenced by the interplay between graph rewriting and monoidal categories, which led in the last

* This work has been supported in part by EPSRC grant EP/P004490/1.

decade to diagrammatic models for quantum computing [1], signal flow [4] and asynchronous circuits [11]. Algebraic specifications in the style of monoidal categories have been pioneered by Sheeran in the 1980s [30] and a certain amount of algebraic reasoning about circuits using such specifications has been attempted [25]. However, a full and systematic categorical presentation of digital circuits has only been given recently by the first two authors of the present paper [13]. Starting only from an equational specification of digital components ("gates") it shows that the free traced monoidal category, subject to certain quotients, is Cartesian. Such categories, known as *dataflow categories* [29, Sec. 6.4] (or *traced cartesian categories* [15]), have very useful equations for iterative unfolding of the trace [6, 15, 31], offering a convenient way to model feedback.

The main theoretical contribution of this paper is providing a rewriting semantics for dataflow categories with a discrete delay operator. It is well known that an algebraic semantics does not automatically translate into an operational semantics, because distributive laws (in particular the functoriality of the tensor product) are directionless. This is where the diagrammatic approach can help when used as a graphical syntax, by avoiding the need for such problematic laws [3] and leading to computationally efficient rewriting. The iteration axioms also raise difficulties, this time of identifying diagrammatic redexes. This problem is compounded by the fact that choosing the wrong iterators to unfold can lead to unproductive rewrites. Finally, the presence of delays raises yet a different set of technical challenges because they cannot be rewritten out of a circuit but only moved around using a *retiming* axiom [24]. We solve these problems by writing circuit diagrams in a particular canonical form, which we call *global trace delay*, for which we can provide effective and efficient unfolding, with certain guarantees of productivity.

The main motivation of this work is to open the door to new optimisation techniques for digital circuits, similar to partial evaluation. We will test our theory against a particularly challenging class of circuits, so called *circuits with combinational feedback* [26]. These are circuits which, despite the presence of feedback loops, behave just like combinational circuits, i.e. they exhibit none of the effects associated with genuine feedback, such as state or oscillation. As is the case with operational semantics, we will see how handling such circuits is mathematically elementary and fully automated. This is indeed remarkable, because the conventional automata-based reasoning method does not accept combinational feedback. Denotational semantics can model such circuits [27] but using rather complex mathematical machinery. Moreover, we will show how circuits with combinational feedback which are parametrised by unspecified "black box" components can be just as easily handled by our approach. As far as we know, there is no existing method for modelling such circuits (called "abstract circuits") in the design literature.

Note. A longer version of this including all proofs is available as a technical report [12].

2 Categorical semantics background

A categorical semantics of circuits was given by the first two authors in an earlier paper [13]. In this section we cover the essential results required to justify the diagrammatic semantics.

2.1 Combinational circuits

Let *object variables*, labelling (collections of) wires, be natural numbers and let *morphism variables* be labels for boxes (e.g., gates and circuits). This is a category of PROducts and Permutations (PROP) [22].

▶ **Definition 1.** Let **Circ** be a categorical signature with objects the natural numbers $\mathbb{N}$ and a finite set of morphisms which may be grouped into the following three classes:

- *levels* (or *values*) $v : 0 \to 1$ forming a lattice $(\mathbf{V}, \sqsubseteq)$;
- *gates* $k : m \to 1$; and
- the special morphisms *join* $\curlyvee : 2 \to 1$, *fork* $\curlywedge : 1 \to 2$, and *stub* $\mathsf{w} : 1 \to 0$.

All circuit signatures include combinators for joining two outputs (*join*) and duplicating an input (*fork*), as well as the ability to discard an output (*stub*). What varies from signature to signature is the number of signal levels and the set of gates. Since levels form a lattice, they must include a smallest element ($\bot$), corresponding to a disconnected input, and a top element ($\top$) corresponding to an illegal output ("short circuit"). In the simplest and most common instance, the set of level has two other elements, *high* and *low*, but it can go beyond that. For example, in the case of metal-oxide-semiconductor field-effect transistors (MOSFET) it makes sense, in certain designs, to model the diode properties of the transistor by taking into account four levels (strong and weak high and low voltage, cf. the relevant IEEE standard for logical simulations [17]).

Circuits, in the sense of this paper, are the morphisms of a free categorical construction over their signature. Beginning with *combinational circuits*, the free construction is as follows:

▶ **Definition 2.** Let **CCirc** be the free symmetric monoidal category over **Circ** and monoidal signature $(\mathbb{N}, +, 0)$, and equations:

Fork: $\curlywedge \circ v = v \otimes v$.

Join: $\curlyvee \circ (v \otimes v') = v \sqcup v'$.

Stub: $\mathsf{w} \circ v = id_0$.

Gate: $k \circ \bigotimes_{i=1,m} v_i = v$, such that whenever $v_i \sqsupseteq v_i'$ then $k \circ \bigotimes_{i=1,m} v_i \sqsupseteq k \circ \bigotimes_{i=1,m} v_i'$.

We will call morphisms in this category *combinational circuits.*

The first three equations model the fact that a fork duplicates a value, a join coalesces two values, and a stub discards anything it receives. The gate equations must cover all possible inputs to a gate k and their particular format entails that the output from a gate is always one of the original levels in $\mathbf{V}$. Since $\mathbf{V}$ is a lattice, the monotonicity requirement is also expressible equationally.

It is known that, in a formal sense, the equality of morphisms in a free SMC corresponds to graph isomorphisms in the diagrammatic language [18], where diagrams are created by the operations of sequential composition ($\circ$), parallel composition ($\otimes$) and symmetry ($\times_{m,n}$, the swapping of two buses with m and n wires, respectively), governed by coherence equations. We will usually write composition in diagrammatical order $f \cdot g = g \circ f$. We write the identity (bus of width m) $\mathrm{id}_m : m \to m$ as simply m. For simplicity we also write $\bigotimes_{i=1,m} f = f^m$, $\bigotimes_{i=1,m} f_i = \mathbf{f}$ and $\bigotimes_{i=1,m} v_i = \mathbf{v}$. For lack of space we will not enumerate the coherence equations here, since they are standard.

The *Gate* axioms state that the behaviour of basic components is fully defined by their inputs, i.e. they are *extensionally complete.* By simple inductive arguments on the structure of morphisms we can establish that all circuits are in fact extensionally complete, i.e. for any circuit (not just gates) $f : m \to n$, for any values $v_i, 1 \leq i \leq m$, there exist unique values $v_j', 1 \leq j \leq n$ such that $f \circ \bigotimes_{i=1,m} v_i = \bigotimes_{i=1,n} v_i'$. Intuitively this means that we only model *local interactions*, abstracting away from global effects such as electromagnetic interference or quantum tunnelling etc.

We can further say that two circuits with the same input-output behaviour are *extensionally equivalent*, and a simple inductive argument shows that this is a *congruence*, i.e. it is an equivalence preserved by sequential and parallel composition. Therefore it makes sense to *quotient* our category **CCirc** and create a new category **ECCirc** in which equivalent circuits are made equal.

ECCirc has interesting additional categorical properties which aid reasoning. Two are of particular importance. The first one is that **ECCirc** is *Cartesian.* The *diagonals* are defined by $\Delta_0 = 0$ and $\Delta_{n+1} = (\Delta_n \otimes \curlywedge) \cdot (n \otimes \mathsf{x}_{(1,n)} \otimes 1)$, *forks* of width n. The diagonal must satisfy two *coherences*: $\langle f, f\rangle = \Delta_n \cdot (f \otimes f) = f \cdot \Delta_m$and $f \cdot \mathsf{w}^m = \mathsf{w}^m$.

Another useful property is that $(\curlywedge, \curlyvee, \mathsf{w}, \perp)$ forms what is known as a *bialgebra*, i.e. an algebraic structure in which $(\curlyvee, \perp)$ is a commutative monoid, $(\curlywedge, \mathsf{w})$ is a co-commutative co-monoid, such that $\curlyvee \cdot \curlywedge = \curlywedge^2 \cdot (1 \otimes \mathsf{x}_{1,1} \otimes 1) \cdot \curlyvee^2$. Finally, fork is a section and join a retraction, $\curlywedge \cdot \curlyvee = 1$, but are not generally isomorphisms. A convenient derived connector is the *join* of width n, defined as $\nabla_0 = 0$ and $\nabla_{n+1} = (n \otimes \mathsf{x}_{1,n} \otimes 1) \cdot (\nabla_n \otimes \curlyvee)$.

2.2 Circuits with discrete delays

▶ **Definition 3.** Let $\mathbf{CCirc}_\delta$ be the category obtained by freely extending **ECCirc** with a new morphism $\delta : 1 \to 1$ subject to the following equations:
Timelessness: For any gate or structural morphism $k : m \to n$, $\delta^m \cdot k = k \cdot \delta^n$.
Streaming: For any gate $k : m \to 1$ and levels $\mathbf{v}$, $(\delta^m \otimes \mathbf{v}) \cdot \nabla_m \cdot k = ((\delta^m \cdot k) \otimes (\mathbf{v} \cdot k)) \cdot \nabla_1$.
Disconnect: $\perp \cdot \delta = \perp$.
Unobservable delay: $\delta \cdot \mathsf{w} = \mathsf{w}$.
Timelessness means that compared to δ, all other basic gates and structural morphisms compute instantaneously. An immediate consequence is that delays can be propagated through combinational circuits, akin to *retiming* [24]. *Disconnect* means that the initial conditions of circuits is $\perp$, so that a wire that also promises to dangle later might as well be considered dangling already. The last rule expresses the same for dangling output wires.

The *Streaming* axiom is more interesting, and it was one of the essential new axioms proposed in [13]. It is key to capturing the intuition of δ as a *delay* operator. Mathematically, first observe that there are infinitely many morphisms of type $0 \to 1$ in $\mathbf{CCirc}_\delta$, not just the finitely many values. This is because expressions such as $v \cdot \delta$ do not reduce to a value. However, it can be shown that any expression built from values, δ, and the structural morphisms can be transformed into a *canonical form* which may be viewed as a *sequence of values presented over time*, something that is called a *waveform* in hardware design lingo. We write a waveform consisting of $n+1$ values as a list $s_n = v_n :: v_{n-1} :: \cdots :: v_0$ where v_n is the value that is currently visible, v_{n-1} becomes visible in the next step, and so on. Formally, $s_0 = v_0$ and $s_{n+1} = (s_n \cdot \delta \otimes v_{n+1}) \cdot \curlyvee$. For example, the expression $v \cdot \delta$ corresponds to the waveform $\perp :: v$; a value v is equal to (any of) the waveforms $v :: \perp :: \cdots :: \perp$ which means that it is only available *now* but no longer in the next time-step. As before, we write $\bigotimes_{i=i,m} s = s^m$ and $\bigotimes_{i=i,m} s_i = \mathbf{s}$.

The *Streaming* axiom now tells us how a gate processes a waveform: we create two separate instances of the gate, one to process the immediate inputs and another to process the subsequent inputs. Applying it repeatedly to a given circuit allows us to determine the waveform that is produced at the output wires. We obtain:

▶ **Theorem 4** (Extensionality). *Given any morphism f in $\mathbf{CCirc}_\delta$, for any input waveform $\mathbf{s}$ there exists a unique output waveform $\mathbf{s}'$ such that $\mathbf{s} \cdot f = \mathbf{s}'$.*

As in the case of circuits without delays, we can show that extensionality is a congruence and we can quotient by it, creating an *extensional* category of circuits with delays, $\mathbf{ECCirc}_\delta$. It is then a routine exercise to show $\mathbf{ECCirc}_\delta$ is Cartesian, with the diagonal and terminal object defined the same way as in **ECCirc**.

2.3 Circuits with feedback

▶ **Definition 5.** Let $\mathbf{CCirc}^*_\delta$ be the category obtained from $\mathbf{ECCirc}_\delta$ by freely adding a trace operator.

Diagrammatically, the trace operator applied to a diagram $f : m + k \to n + k$ corresponds to a feedback loop of width k, written $\mathrm{Tr}^k(f) : m \to n$. Symmetric traced monoidal categories (STMC) satisfy a number of equations (coherences) which we will not enumerate for lack of space [19]. As before, their interpretation coincides with equality of diagrams (with feedbacks) up to graph isomorphism.

As before, we are committed to an extensional view of circuits where the only observable is the input-output behaviour. In combinational circuits, with or without delays, the only way we can create a circuit with 0 outputs is by explicitly composing a circuit $f : m \to n$ with w^n. However, 0-output circuits can arise in more complicated ways in the presence of feedback, whenever all the outputs are fed back. It is convenient and reasonable to equate all 0-output circuits to w^n, trivially a congruence. The new quotient category is called $\mathbf{OCirc}^*_\delta$. In this category all diagrams of shape $f : m \to 0$ are therefore equal which, categorically speaking, makes 0 a "*terminal object*".

In general, in programs feedback corresponds to recursion and iteration, and syntactic models (operational semantics) of such programs involve creating two copies of the code recursed over. For example, the operational semantics of the Y-combinator as applied to some G is $YG = G(YG)$. A similar rule does not exist in general for SMTCs unless the category is also Cartesian. Such categories, also called *data-flow categories* [6], admit an *iterator* defined for any $f : m + n \to n$, $\mathrm{iter}^n(f) = \mathrm{Tr}^n(f \cdot (\Delta_n \otimes n)) : m \to n$, which satisfies
Naturality : $\mathrm{iter}((g \otimes n) \cdot f) = g \cdot \mathrm{iter}(f)$ for any $g : k \to m$,
Iteration : $\mathrm{iter}(f) = \langle m, \mathrm{iter}(f)\rangle \cdot f$
Diagonal : $\mathrm{iter}^n(\mathrm{iter}^n(f)) = \mathrm{iter}^n((\langle n, n\rangle \otimes m) \cdot f)$.

We can use these equations because the category in which we operate is indeed Cartesian.

▶ **Theorem 6** ([13]). *The category* $\mathbf{OCirc}^*_\delta$ *is Cartesian with diagonal* Δ_n.

To conclude the section, we discuss the existence of a concrete model for $\mathbf{OCirc}^*_\delta$ which will confirm the axiomatic framework is consistent. It needs to be Cartesian and support the delay operator and iteration. The usual example of a traced SMC, sets and relations, is not Cartesian so a slightly more complex construction is required. We start with a basic model for combinational circuits based on the lattice $\mathbf{V}$ of values (Def. 1).

▶ **Definition 7.** Let $\mathcal{V}$ be the category whose objects are finite powers of $\mathbf{V}$ and whose morphisms are monotone maps.

▶ **Theorem 8** ([12]). *There is a unique traced monoidal functor* $[\![\cdot]\!]^{\mathcal{S}}$ *from* $\mathbf{CCirc}^*_\delta$ *to* $\mathcal{S}$ *mapping the object* 1 *of the former to* $\mathbf{V}$ *of the latter.*

3 Diagrammatic operational semantics

The results of the previous section establish a powerful framework for algebraic reasoning about circuits. However, this framework is not equally useful for *automatic* reasoning and cannot implement a reasonable operational semantics.

The first obstacle is the functoriality property of the tensor, which lacks directionality. Consider the circuit corresponding to the boolean expression $t \wedge f \wedge t$, where the constants involved satisfy the obvious equations. This diagram can be specified in several ways. Some of the specifications, e.g. $(((t\otimes f)\cdot\wedge)\otimes t)\cdot\wedge$ have the immediately identifiable redex $(t\otimes f)\cdot\wedge = f$ which reduces the overall expression to $(f\otimes f)\cdot\wedge$, which reduces to f. However, the same circuit can be equivalently written as $(t\otimes f\otimes t)\cdot(\wedge\otimes id)\cdot\wedge$ which has no obvious redex. Finding redexes in such structural diagram specifications is computationally prohibitive and an unsuitable operational semantics. The alternative is to exploit the connection between monoidal categories in general, and traced monoidal categories in particular, and certain graphs. This idea has been analysed in depth recently [3].

We will give a concrete presentation of the graphs following Kissinger's *framed point graphs*, which are a free (strict) symmetric traced monoidal category [20, Thm. 5.5.10]. To make the presentation more accessible we will elide some of the categorical technicalities in *loc. cit.* and give a more direct presentation.

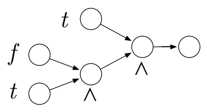
Let a labelled directed acyclic graph (LDAG) be a DAG (V, E) equipped with a partial labelling function $f : V \rightharpoonup L$. Let a labelled interfaced DAG (*LIDAG*) be a labelled DAG with two distinguished lists of unlabelled nodes representing the “input" (I) and “output" (O) interfaces, $G = (V, E, f, I, O)$. We write the set of elements of a list L as $|L|$, and list concatenation and cons both as $- :: -$. Let the *zip* operation on lists be defined as usual, $zip\ nil\ nil = nil$, and $zip\ x :: xs\ y :: ys = (x, y) :: zip\ xs\ ys$.

Unlabelled nodes are called *wire nodes* and edges connecting them are called *wires*. A *wire homeomorphism* [20, Sec. 5.2.1] is any insertion or removal of wire nodes along wires, which does not change the shape of the graph. Two *LIDAG*s are considered to be equivalent if they are graph isomorphic up to renaming vertices and wire homeomorphisms:

$$(V \uplus \{a, b, c\}, E \uplus \{(a, b), (b, c)\}, f, I, O) \simeq (V \uplus \{a, c\}, E \uplus \{(a, c)\}, f, I, O),$$

if and only if $f(b)$ is undefined. The quotienting of *LIDAG*s by this equivalence gives us *framed point graphs* (FPG) [20, Def. 5.3.1].

The algebraic specifications of the diagrams associated with the expression $t \wedge f \wedge t$ mentioned above all correspond to the (same) framed point graph with empty input interface and 1-point output interface:

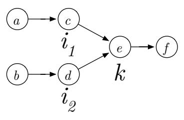


This representation solves the problem raised by the functoriality of the tensor, as redexes can be detected in linear time in the size of the graph. Note that all wire nodes can also be removed in linear time in terms of the size of the graph.

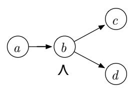
Sequential composition of two FPGs where the size of the output of the first matches the size of the input of the second is defined by identifying the output list and the input list of the two graphs. Since FPGs are equal up to renaming of vertices, the names of the wires can be chosen so that the composition is well defined. The unlabelled input and output nodes become wire nodes in the composition. The tensor is the disjoint union of the two graphs. It is always well defined since graphs are identified up to vertex renaming. The trace operator picks the head nodes of the input and output lists of points, makes them wire nodes, and connects them. Formally, if $G_i = (V_i, E_i, f_i, I_i, O_i)$ and $G = (V, E, f, a :: I, b :: O)$ then

$$G_1; G_2 = (V_1 \uplus V_2, E_1 \uplus E_2 \uplus |zip\ O_2\ I_1|, f_1 \uplus f_2, I_1, O_2)$$
$$G_1 \otimes G_2 = (V_1 \uplus V_2, E_1 \uplus E_2, f_1 \uplus f_2, I_1 :: I_2, O_1 :: O_2)$$
$$\mathrm{Tr}(G) = (V, E \uplus \{(b, a)\}, f, I, O).$$

Constants are interpreted by the graphs below:



$k : 2 \to 1$



$\curlywedge : 1 \to 2$

A binary gate is shown, but n-ary gates and *join* are similar. The labels i_1, i_2 are required to identify inputs on non-commutative constants but can be otherwise omitted (e.g. in the case of join $\curlyvee : 2 \to 1$). If unambiguous we shall not display the node identities $(a, b, \ldots)$ just their labels, if any.

The graph representation provides a solution for dealing with the functoriality of the tensor, but the presence of feedback raises a new, additional problem. Suppose that we deal with a graph which includes several iterations, e.g. $\text{iter}(f) \cdot \text{iter}(g)$. This graph raises two computationally difficult questions. The first one is how we identify feedback patterns efficiently so that we can apply the iteration axiom. The second one is, if there are several instances of the iteration unfolding axiom that can be applied, what is the schedule of applying them? Without a good (linear time) solution to the first problem we cannot claim that we have a genuine operational semantics. Without a good solution to the second problem we run into technical problems of termination and confluence. Diagrammatic representation alone is no longer the solution.

The main contribution of this section, and of the paper, is showing how to solve these two problems.

Before we proceed, we will give a generalisation of the Streaming axiom which will aid the formulation of the diagrammatic semantics, which relies in turn on a general property which holds in all free symmetric monoidal categories which we call *staging*.

▶ **Lemma 9** (Staging). *Given a free SMC over a signature Γ, any morphism f can be written as a sequence of compositions $f = f_0 \cdot f_1 \cdots f_n$ where f_i is a tensor including exactly one non-identity morphism, $f_i = m \otimes k \otimes n$.*

Let us call *passive* a circuit which has no occurrences of a value. Using the Staging Lemma (9) we can show that:

▶ **Lemma 10** (Generalised Streaming). *For any passive combinational circuit $f : m \to n$, $(\delta^m \otimes m) \cdot \nabla_m \cdot f = (f \cdot \delta^n \otimes f) \cdot \nabla_n$.*

Moreover, a diagram with feedback loops can always be rewritten as single, global, feedback loop.

▶ **Lemma 11** (Global trace form). *For any morphism f in a free STMC there exists a trace-free morphism $\hat{f}$ such that $Tr^n(\hat{f}) = f$ for some $n \in \mathbb{N}$.*

This form can be maintained in the graph representation with constant overhead. In the graph we can maintain a distinguished subset of known *feedback* wire nodes so that the feedback loops can be immediately identified. This can be done compositionally just by keeping track of the feedback wire nodes in sequential composition, tensor and trace. By maintaining the feedback wire nodes explicitly we can ensure two useful invariants. First, the rest of the graph is a DAG. Second, for each feedback wire node there is precisely one incoming and one outgoing edge. We call these graphs *trace-framed point graphs* (TFPGs). Note that feedback wire nodes must not be entirely removed as wire homeomorphisms are applied. Feedback edges that bypass the set of feedback wire nodes are legal, but break the TFPG form. Maintaining these restrictions is computationally trivial (constant overhead).

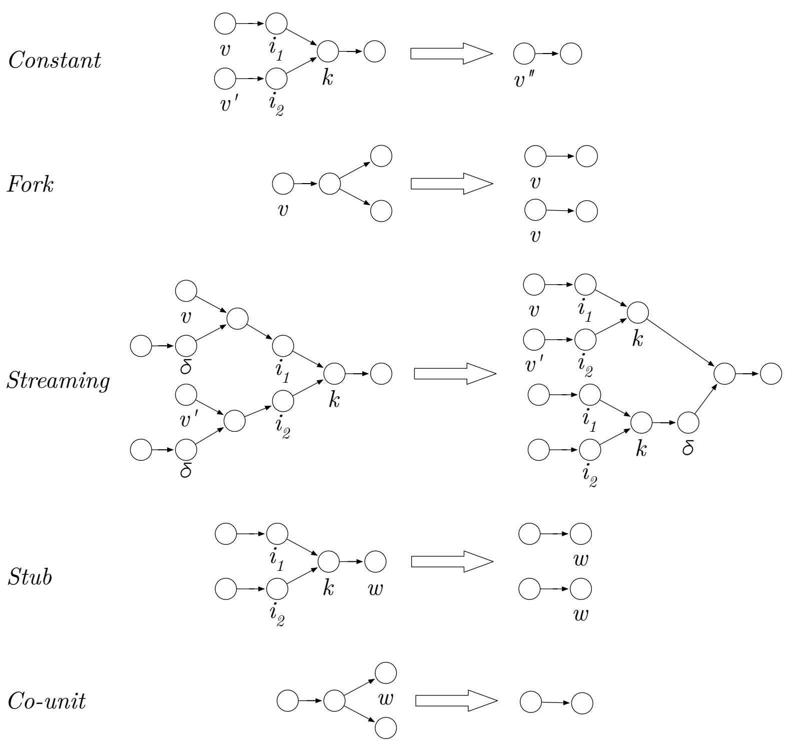


Figure 1 Rewrite rules for combinational circuits.

We are now in a position to define the diagrammatic semantics as a graph-rewriting system in which each rule can be applied efficiently, in linear time as a function of the size of the graph.

3.1 Rewrite rules for combinational circuits

The categorical equations can be expressed as graph rewrite rules, summarised in Fig. 1. We give the rules in an informal diagrammatic style, but a formalisation in an established formalism such as DPO [8] is a standard exercise.

The *Constant* rule shown is for binary constants, but rules for constants of different arity are similar. In the case of the *Constant* rule we require $v'' = (v \otimes v') \cdot k$.

Enhanced Constant Rules. Besides the basic equations for constants, more equations can be proved by extensionality in which reductions can be carried out without all input values being present. For example, $\mathit{true} \vee x = \mathit{true}$ or $\mathit{true} \wedge x = x$. These equations are admissible in the rewrite system.

We call the rewrite rules above the *local rewrite rules.* A TFPG where no local rewrite rules apply is in *canonical form.* The following basic properties of the rewrite system hold.

▶ **Proposition 12.** *The local rewrite rules are sound relative to the categorical equations.*

▶ **Lemma 13.** *The local rewrite rules are strongly confluent.*

▶ **Lemma 14** (Progress). *A circuit $f : 0 \to n, n \neq 0$ without traces or delays is either a value or the TPFG associated with it has redexes.*

From Lem. 13 and 14 it follows that

▶ **Theorem 15.** *Given a circuit $f : 0 \to m, m \neq 0$ in* **ECCirc** *the local rules will always rewrite its TPFG representation in a finite number of steps into a TPFG representation of a **value*** $\mathbf{v}$ *such that $f = \mathbf{v}$.*

Finally, it can be shown that the graph rewriting is *efficient*.

▶ **Lemma 16.** *Any rule of the graph rewrite system is applicable in linear time (in the size of the graph).*

3.2 Feedback and delay

We now need to add rules for delays, which may occur in arbitrary places in the circuit, not just in waveforms. For example, a circuit such as $(t \otimes f) \cdot (1 \otimes \delta) \cdot \wedge$, in TFPG representation, does not have any redex because of the delay. Dealing with the delays requires a complex rule which takes into account the presence of the trace. The trace and the delay must be dealt with together because of the following result which allows us to write any circuit in what we will call *global-delay form*. Note Lem. 10 does not hold for combinational circuits with values. However, the following holds:

▶ **Lemma 17.** *For any combinational circuit $f : m \to n$ there exists a passive circuit $\tilde{f}$ such that $f = (m \otimes \mathbf{v}) \cdot \tilde{f}$ for some $\mathbf{v}$.*

We call the application of the transformation in this lemma the *passification* of the circuit.

▶ **Lemma 18.** *Any circuit f in* $\mathbf{OCirc}^*_\delta$ *can be written as $f = \mathrm{Tr}^m((\delta^n \otimes p) \cdot f')$ for some trace-free, delay-free circuit f', $m, n, p \in \mathbb{N}$.*

Trace-Delay. The most complex rule is the unfolding of the global trace, which also handles the delays. First, we need an unfolding axiom for trace from the unfolding axiom for iteration, by expressing trace in terms of iteration. We have seen that the iterator can be expressed in term of trace, but the converse is also possible.

▶ **Proposition 19** ([15])**.** *For any $f : A \otimes X \to A \otimes Y$,*

$$\mathrm{Tr}^A(f) = \mathrm{iter}^{A \otimes Y}((id_A \otimes \mathsf{w}_Y \otimes id_X) \cdot f) \cdot (\mathsf{w}_A \otimes id_Y).$$

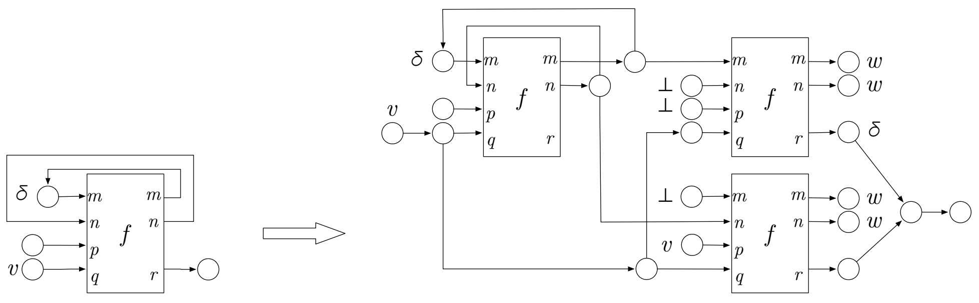
Routine calculations give the following formula for unfolding the trace operator:

▶ **Proposition 20.** *For any $f : A \otimes X \to A \otimes Y$,*

$$\mathrm{Tr}^A(f) = \Delta_X \cdot (\mathrm{iter}^A(f \cdot \mathsf{w}_Y) \otimes id_X) \cdot f \cdot (\mathsf{w}_X \otimes id_Y).$$

Using the above, and the fact that any circuit can be written as a passified (Lem. 17), global-trace, global-delay circuit we can give the following global rewrite rule for circuits with feedback and delays.

▶ **Proposition 21.** *Given a graph representing a passified, global trace, global delay circuit, $f : m + n + p + q \to m + n + r$, the following rewrite rule is sound:*



Proof (Sketch). This rewrite rule is a sequence of valid rewrites. Step (1) represents the unfolding of the trace (Prop. 20). Step (2) uses $\perp$ as the unit of the join-monoid along with the *Unobservable Delay* axiom, to bring the circuit to a form where *Generalised Streaming* (Lem. 10) can be applied, which is step (3). A final simplification removes redundant delays which are not observable (step (5)). A final step (6) restore the global-delay form, using Lem. 18. The resulting circuit can be represented as a TFPG. ◀

The unfolding rule is also efficient:

▶ **Lemma 22.** *A passified, global-trace, global-delay circuit can be unfolded in a time linear in the size of its graph representation.*

We define the overall rewriting system as a cycle of local rewrites until canonical form is reached, followed by trace-delay unfoldings. This system is obviously not terminating, which is consistent with the fact that circuits with feedback can generate infinite waveforms. E.g., $\mathrm{iter}(v :: 1) = v :: \mathrm{iter}(v :: 1) = v :: v :: \mathrm{iter}(v :: 1) = \cdots$.

3.3 Productivity

In a circuit of the form $v :: f = (v \otimes (f \cdot \delta)) \cdot \curlyvee$ value v will be observed *before* whatever the behaviour of f is, since v is *instantaneous* whereas f is guarded by a delay. We call such circuits *productive*, and we add a *labelled* rewrite rule to simplify productive circuit by removing the produced value: $v :: f \overset{v}{\Longrightarrow} f$. This rule is sound because the sub-circuit $v :: -$ can never be part of any redex. So the example above can be written as: $\mathrm{iter}(v :: 1) = v :: \mathrm{iter}(v :: 1) \overset{v}{\Longrightarrow} \mathrm{iter}(v :: 1)$.

However, we note circuits need not be productive in general. There exist circuits where unfoldings never reduce to shape $v :: f$, e.g. $t \cdot \mathrm{iter}(\wedge)$. This is a well known problem caused by a genuine *instant feedback* loop between the output and one of the inputs of the gate. If a circuit has no instant feedback loops, it is guaranteed to be productive.

▶ **Definition 23.** We say that a circuit has *delay-guarded feedback* if its global-delay form is $\mathrm{Tr}^m(\delta^m \cdot f)$.

If a circuit has delay-guarded feedback loops then it is productive. In fact it implements a Mealy automaton.

▶ **Theorem 24.** *Closed delay-guarded circuits with no inputs are productive. Given the TPFG representation of a delay-guarded feedback, the rewrite system will produce a TPFG graph representing a circuit $v :: g$ in a finite number of steps.*

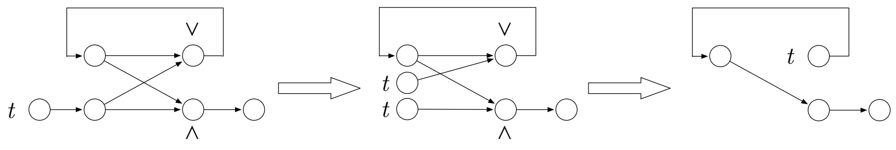
By *closed* above we mean that all inputs to the circuit are provided, i.e. it has type $0 \to m$. Note that the delay-guarded feedback condition is sufficient but not necessary. An interesting example of circuits with non-delay guarded feedback which are productive are the cyclic combinational circuits which we discuss below.

To be able to use the diagrammatic semantics as an operational semantics, we also give a necessary and sufficient non-productivity criterion.

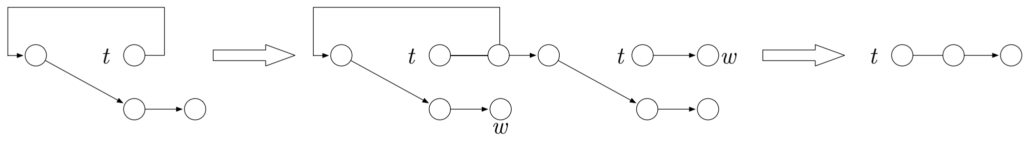
▶ **Theorem 25.** *If a closed, global-trace, global-delay circuit is unproductive after one unfolding then it will always be unproductive.*

3.4 Example: Cyclic combinational circuits

A challenging class of circuits, which are rejected by standard digital design tools, are combinational circuits with feedback which is not delay guarded [26]. Consider Boolean circuits with *and* and *or* gates. Below is an example of such a circuit. Closing the circuit by applying a boolean value at the input (e.g. t) makes it possible to apply the diagrammatic semantics, using the enhanced equational rewrite rules:



There is no rule for "yanking" the superfluous trace, but unfolding the diagram again achieves the same purpose. The circuit then reduces to the constant t, by applying the co-unit and *stub* rules.

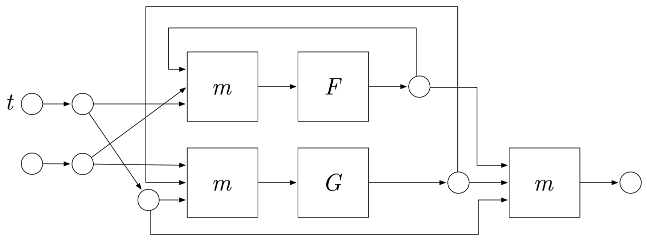


4 Specialising open abstract digital circuits

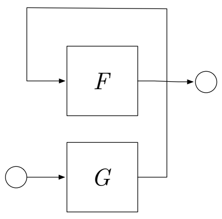
If we are not using the rewrite rules as an operational semantics, and so are not concerned with productivity issues, we can apply the reduction rules to open and to "abstract" circuits, with unspecified components. This gives us a basis for powerful partial evaluation-like optimisations of circuits. This is a new contribution with potentially interesting practical applications.

Consider the circuit represented by the TFPG below, where the gate $m : 3 \to 1$ is a *multiplexer* and F, G are abstract circuits. For readability we omit the input labels of the multiplexer. This circuit, presented in [26], implements the operation *if* x *then* $F(G(y))$ *else* $G(F(y))$. The circuit has no delays so the feedback loops are combinational, so they are rejected by conventional circuit analysis tools, which disallow instant feedback. However, the multiplexers are set up so that no matter what the value applied at x, the residual circuit is feedback-free. The false feedback loops in the circuit are only a clever way to reuse the two abstract circuits F and G.

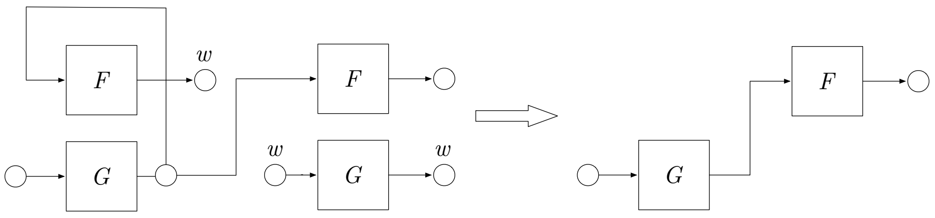
Consider the case when x becomes t, and y is left unspecified:



Routine repeated application of the local rewrite rules for fork, m, and stub results in a circuit which still has a residual feedback loop:



This feedback loop can be yanked, and the circuit is just $G \cdot F$. However, the system does not have a *yank* rule as it would be too expensive to implement, so the unfolding rule is applied again! The *Stub* rule will then remove the first occurrence of F and the second occurrence of G, resulting, as expected, in $G \cdot F$.



To conclude, we would like to emphasise how a circuit that poses a triple challenge to standard digital design tools (open, abstract, combinational feedback) can be partially evaluated in completely automated fashion by our diagrammatic semantics, resulting in a much simpler specialised circuit.

4.1 Pre-logical circuits

We can also model operationally transistor-level circuits, which is also a new capability afforded by the diagrammatic semantics. The circuit framework is general enough to allow operational reasoning about digital circuits at a level of abstraction below logical gates, for example metal-oxide-semiconductor field-effect transistor (MOSFET) circuits. In *saturation mode* such transistors can be considered to take on a discrete set of values which, depending on the circuit and the analysis, can be four-valued (*high impedance* $<$ *high, low* $<$ *unknown.*) or six-valued (*high impedance* $<$ *weak high, weak low*; *weak high* $<$ *strong high*; *weak low* $<$ *strong low*; *strong high, strong low* $<$ *unknown.*). Unlike Boolean logic, where the wire-join construct is not used, in a transistor circuit output wires are joined, and the semantics of the wire-join is that of the value-lattice join operator.

We will work in the six-value lattice $\bot$ (high impedance), h (weak high), H (strong high), l (weak low), L (strong low), $\top$ (unknown). We will take the (idealised) nMOS and pMOS transistors as the basic gates. The nMOS transistor ($\mathsf{n} : 2 \to 1$) works like a low-activated switch, but it only allows low current to flow. High current can flow, but is much diminished. The behaviour of the transistors can be defined equationally in this setting. When implementing a logical gate in MOSFET we want H to correspond to *true* and L to false. The correct behaviour of a gate must keep this representation without, e.g. producing $\top$ or weak output h, l.

Let us now revisit the example of the previous section, but with the multiplexer implemented down to transistors. A very simple circuit is the *inverter*, with which we can build a

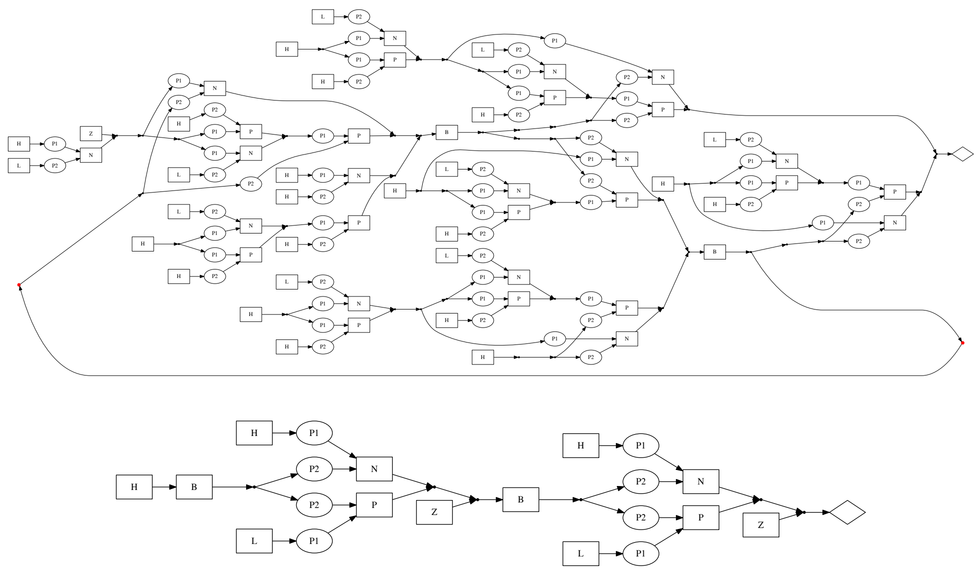


Figure 2 MOSFET circuit and the residual circuit after partial evaluation.

pass-through gate (pass), and the multiplexer (m):

$$\begin{aligned}
\mathsf{inv} &= \curlywedge \cdot (1 \otimes \mathsf{h} \otimes 1 \otimes \mathsf{l}) \cdot (\mathsf{p} \otimes \mathsf{n}) \cdot \curlyvee \\
\mathsf{pass} &= \curlywedge^2 \cdot (\mathsf{inv} \otimes 3) \cdot (1 \otimes \mathsf{x} \otimes 1) \cdot (\mathsf{p} \otimes \mathsf{n}) \cdot \curlyvee \\
\mathsf{m} &= (\curlywedge \otimes 2) \cdot (1 \otimes \mathsf{x} \otimes 1) \cdot (2 \otimes \mathsf{inv} \otimes 1) \cdot \mathsf{pass}^2 \cdot \curlyvee .
\end{aligned}$$

The abstract circuit from the previous section is represented as a TFPG in the first graph in Fig. 2, and is specialised relative to the abstract circuits (denoted as B in the graph) using a prototype tool[1]. In this case both inputs are provided. The residual circuit is shown as the second graph. It is interesting to note that the MOSFET version of the circuit leads to a different residual circuit compared to the more high level circuit of the previous section. The reason is that reducing the pass-through gates would require more complex rewriting, which cannot be done efficiently in general.

5 Conclusion, related and further work

Some theoretical ingredients we have used in this work have been around for quite a while and it is perhaps somewhat surprising that they have not been put together for a coherent operational and diagrammatic treatment of digital circuits. Our Thm. 6 implies that $\mathbf{OCirc}^*_\delta$ is a Lawvere theory [9] with trace, also known as an iteration theory [10], a concept which has been studied extensively [2], leading to recent connections with rewrite systems [14]. The relation between trace and iteration has also been studied before in a somewhat similar categorical setting [16]. The connection between Lawvere theories and PROPS has also been recently studied [5].

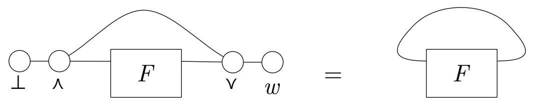
It is also quite surprising that despite major early progress in the algebraic treatment of circuits, [30, 25], this line of work has not come earlier to a systematic conclusion. But the

[1] `https://github.com/AliaumeL/circuit-syntax`

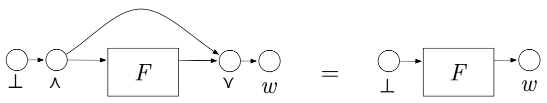
contribution of our work is not merely assembling off-the-shelf components. The *Streaming* axiom is new, and the fact that it generalises to arbitrary passive combinational circuits is a crucial ingredient for our work. To make the unfolding of iteration computationally tractable, the diagrammatic representation required a non-obvious canonical form, which must be easy to compute. Without it our earlier semantics [13] cannot be used as an effective operational semantics.

We have been in particular inspired by the deep connections between monoidal categories and diagrams [29] which *inter alia* have been used in the modelling of quantum protocols [1] and signal-flow graphs [4]. Some contrasts are quite interesting. Unlike in quantum protocols, all digital circuits with no inputs and no outputs are equal whereas in quantum computing they correspond to *scalars*, which allow quantitative aspects to be expressed. Should we have taken a similar direction we could have included quantitative aspects such as power consumption in our formalism, but we would have lost the diagonal property. Obviously, two copies of a circuit will at least sometimes consume more power than one copy.

The signal-flow graphs in [4] are linear and reversible, which is not the case for digital circuits. Without elaborating the mathematics too much, a key difference between their model and ours can be illustrated by the following equality, involving the interaction between fork, join, and disconnected wires, as a trace can be created out of a fork and a join:



Of course, by comparison, in our setting the directionality of the wires never changes, so the correct equality for us is, in contrast:



These two simple diagrammatic equations above truly capture the essential difference between *electric* and *electronic* circuits!

Beyond the scholarly context and technical innovations, we are most excited about the potential applications of our work. Cyclic combinational circuits are a litmus test for circuit modelling theories and we hope the reader can appreciate that in our framework their model is elementary. For comparison, there are few theories that can handle such circuits, and they demand a significant level of mathematical sophistication [27]. The true potential of our method should be the unleashing of symbolic, operational and syntactic methods, such as partial evaluation, for reasoning about and optimising circuits, methods which proved so effective in programming languages.

Acknowledgements. We thank George Constantinides and Alex Smith for feedback and suggestions.

References

1 Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *LICS*, pages 415–425, 2004.

2 Stephen L. Bloom and Zoltán Ésik. *Iteration Theories: The Equational Logic of Iterative Processes.* Springer-Verlag New York, Inc., New York, NY, USA, 1993.

3 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In *LICS*, pages 710–719, 2016.

4 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Full abstraction for signal flow graphs. In *POPL*, pages 515–526, 2015.

5 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Lawvere categories as composed props. In *CMCS*, pages 11–32, 2016.

6 Virgil Emil Căzănescu and Gheorghe Ştefănescu. Towards a new algebraic foundation of flowchart scheme theory. *Fund. Inf.*, 13(2):171–210, 1990.

7 Charles Consel and Olivier Danvy. Tutorial notes on partial evaluation. In *POPL*, pages 493–501, 1993.

8 Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. Algebraic approaches to graph transformation – part i: Basic concepts and double pushout approach. In *Handbook of Graph Grammars*, pages 163–246, 1997.

9 Samuel Eilenberg and Jesse B. Wright. Automata in general algebras. *Information and Control*, 11(4):452–470, 1967.

10 Calvin C. Elgot. Monadic computation and iterative algebraic theories. *Studies in Logic and the Foundations of Mathematics*, 80:175–230, 1975.

11 Dan R. Ghica. Diagrammatic reasoning for delay-insensitive asynchronous circuits. In *Computation, Logic, Games, and Quantum Foundations*, pages 52–68, 2013.

12 Dan R. Ghica, Achim Jung, and Aliaume Lopez. Diagrammatic semantics for digital circuits. In *Quantum Physics and Logic (QPL)*, 2017. forthcoming.

13 D. R. Ghica and A. Jung. Categorical semantics of digital circuits. In *Formal Methods in Computer-Aided Design (FMCAD)*, 2016.

14 Makoto Hamana. Strongly normalising cyclic data computation by iteration categories of second-order algebraic theories. In *FCSD*, pages 21:1–21:18, 2016.

15 Masahito Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of let and letrec*. Springer Verlag, 1999.

16 Masahito Hasegawa. The uniformity principle on traced monoidal categories. *Electr. Notes Theor. Comput. Sci.*, 69:137–155, 2002.

17 IEEE model standards group. IEEE standard multivalue logic system for VHDL model interoperability (std_logic_1164), May 1993. `doi:10.1109/IEEESTD.1993.115571`.

18 André Joyal and Ross Street. The geometry of tensor calculus, I. *Adv. in Math.*, 88(1):55–112, 1991.

19 André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. In *Math. Proc. of the Cambridge Phil. Soc.*, volume 119, pages 447–468. Cambridge Univ. Press, 1996.

20 Aleks Kissinger. *Pictures of Processes*. PhD thesis, University of Oxford, 2011. arXiv:1203.0202v2.

21 Robert P. Kurshan and Kenneth L. McMillan. Analysis of digital circuits through symbolic reduction. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 10(11):1356–1371, 1991.

22 Stephen Lack. Composing PROPs. *Theory and App. of Categories*, 13(9):147–163, 2004.

23 Peter J. Landin. An abstract machine for designers of computing languages. In *Proc. IFIP Congress*, volume 65, 1965.

24 Charles E. Leiserson and James B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1-6):5–35, 1991.

25 Wayne Luk. Pipelining and transposing heterogeneous array designs. *J. of VLSI Sig. Proc. Sys.*, 5(1):7–20, 1993.

26 Sharad Malik. Analysis of cyclic combinational circuits. In *Proc. IEEE/ACM Int. Conf. on Comp. Aided Design*, pages 618–625, 1993.

27 Michael Mendler, Thomas R. Shiple, and Gérard Berry. Constructive boolean circuits and the exactness of timed ternary simulation. *Form. Meth. Syst. Des.*, 40(3):283–329, 2012.

28 Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

29 Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.

30 Mary Sheeran. muFP, A language for VLSI design. In *LISP and Func. Prog.*, pages 104–112, 1984.

31 Alex Simpson and Gordon Plotkin. Complete axioms for categorical fixed-point operators. In *Logic in Computer Science, 2000. Proceedings. 15th Annual IEEE Symposium on*, pages 30–41. IEEE, 2000.