Diagrammatic Semantics for Digital Circuits

Dan Ghica and **George Kaye**

27 November 2020

University of Birmingham

Introduction

We normally reason about hardware by using simulation.

But what about syntactic reasoning?

We can develop an alternative approach – an operational semantics for digital circuits!

We take an approach rooted in monoidal categories.

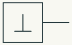
Building circuits categorically

Combinational circuits

Circuits are defined as morphisms ('building blocks') in a symmetric traced monoidal category.

Values forming a lattice

$$\bot : 0 \to 1$$



$$t : 0 \to 1 \qquad f : 0 \to 1$$

$$\top : 0 \to 1$$

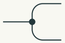
Gates

$$AND : 2 \to 1 \qquad m : 3 \to 1$$

Structural morphisms

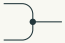
$\curlywedge : 1 \to 2$

Fork



$\curlyvee : 2 \to 1$

Join

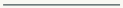


$\sim : 1 \to 0$

Stub

$n : n \to n$

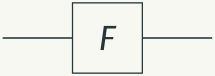
Identity

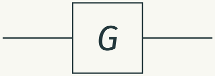


Sequential composition

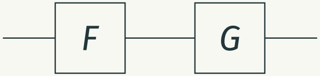
We can compose circuits sequentially...

$F : m \to n$ $\qquad$ $G : n \to p$





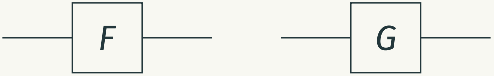
$F \cdot G : m \to p$



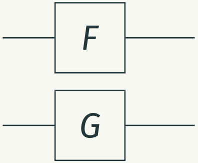
Parallel composition

...or in parallel!

$$F : m \to n \qquad G : p \to q$$



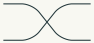
$$F \otimes G : m + p \to n + q$$



Symmetry

Wires can be swapped using a symmetry.

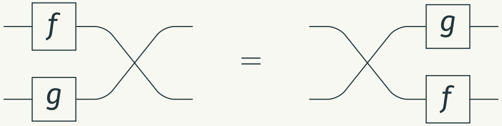
$$\times_{m,n} \; : \; m + n \; \to \; n + m$$



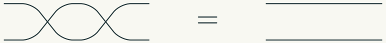
Satisfying some axioms...

Symmetry – axioms

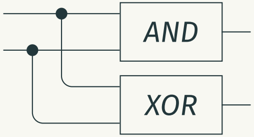
Naturality



Self-inverse



Example – half adder



$$\mathbf{1} \otimes \curlywedge \cdot \curlywedge \otimes \mathbf{2} \cdot \mathbf{1} \otimes \times_{\mathbf{1},\mathbf{1}} \otimes \mathbf{1} \cdot \mathit{AND} \otimes \mathit{XOR}$$

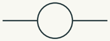
Already getting quite hard to read!

Delay

So far everything has been combinational.

We represent delay using a special morphism δ_t, indexed by elements of a monoid $t \in \mathbf{T}$.

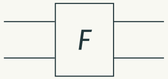
$$\delta_t \ : \ \mathsf{1} \to \mathsf{1}$$



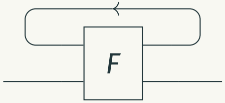
Feedback

We can represent feedback using a trace.

$$F \;:\; x + m \;\to\; x + n \qquad\qquad \mathrm{Tr}^{x}(F) \;:\; m \;\to\; n$$



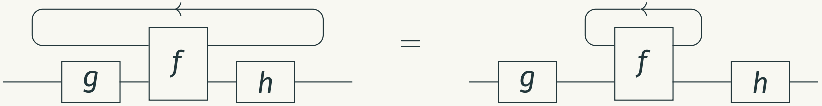
$\Rightarrow$



Feedback – axioms

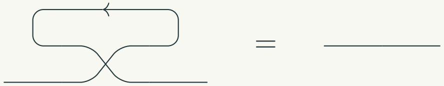
Tightening

$$\mathrm{Tr}^x(x \otimes G \cdot F \cdot x \otimes H) = G \cdot \mathrm{Tr}^x(F) \cdot H$$



Yanking

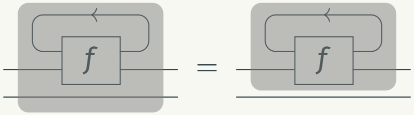
$$\mathrm{Tr}^x(\times_{x,x}) = x$$



Feedback – axioms

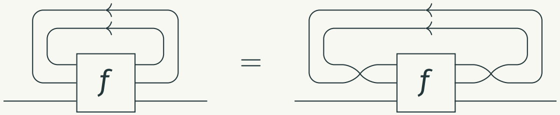
Superposing

$$\mathrm{Tr}^x(F \otimes m) = \mathrm{Tr}^x(F) \otimes m$$

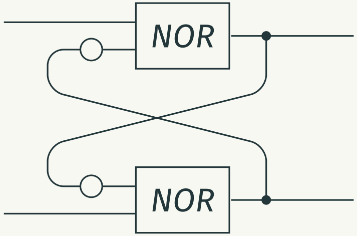


Exchange

$$\mathrm{Tr}^y(\mathrm{Tr}^x(F)) = \mathrm{Tr}^x(\mathrm{Tr}^y(\times_{y,x} \otimes m \cdot F \cdot \times_{x,y} \otimes n))$$



Example – flipflop



$$\mathrm{Tr}^{1}((\times_{\mathbf{1},\mathbf{1}} \cdot \mathbf{1} \otimes \delta \cdot \mathit{NOR} \cdot \curlywedge) \otimes \mathbf{1} \cdot \mathbf{1} \otimes (\delta \otimes \mathbf{1} \cdot \mathit{NOR} \cdot \curlywedge) \cdot \times_{\mathbf{1},\mathbf{1}} \otimes \mathbf{1})$$

This makes very little sense now...

Reasoning about circuits

We have now defined the structure of our circuits.

To define the behaviour of our circuits, we will have to introduce more axioms.

For example, $\mathsf{t} \otimes \mathsf{t} \cdot \mathit{AND} = \mathsf{t}$.

Reasoning about circuits

We can identify redexes and reduce them appropriately.

$$(\mathbf{t} \otimes \mathbf{t} \cdot \mathbf{AND}) \otimes (\mathrm{t} \otimes \mathrm{t} \cdot \mathit{AND}) \cdot \mathit{AND}$$

$$\mathrm{t} \otimes (\mathbf{t} \otimes \mathbf{t} \cdot \mathbf{AND}) \cdot \mathit{AND}$$

$$\mathbf{t} \otimes \mathbf{t} \cdot \mathbf{AND}$$

$$\mathrm{t}$$

Great! But what about something like

$$(\mathrm{t} \otimes \mathrm{t} \otimes \mathrm{t} \otimes \mathrm{t}) \cdot (\mathit{AND} \otimes \mathit{AND}) \cdot \mathit{AND}$$

Where have the redexes gone?

We need to find an efficient way to identify redexes.

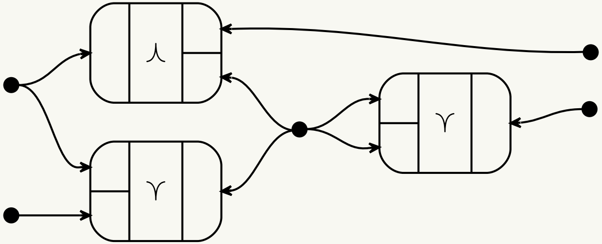
The diagrammatic avenue

We have been using these informal diagrams to illustrate our term language.

What if we were to formalise these diagrams into a sound and complete diagrammatic language?

Hypergraphs

Hypergraphs



Hyperedges (boxes) have sources (left) and targets (right).

Arrows represent the incidence of vertices on edges.

Vanilla hypergraphs are not enough

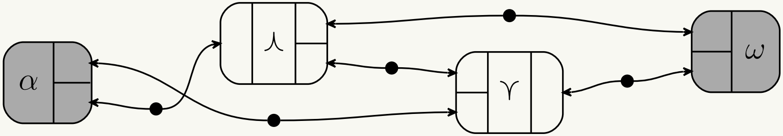
Vertices in hypergraphs can connect to any number of edges.

But in our circuits we need to use an explicit combinator to split or merge wires.

We also need to keep track of the inputs and outputs (the interface) of our hypergraph.

To fix these problems we introduce a restriction on simple hypergraphs that we call linear hypergraphs.

Linear hypergraphs

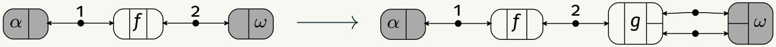


Each vertex has a ‘left’ and a ‘right’ connection.

Special edges α (for inputs) and ω (for outputs).

Linear hypergraph homomorphisms

We can embed a subgraph F into another graph G by using a hypergraph homomorphism.



If the maps between edges and vertices are bijective then F and G are isomorphic $F \equiv G$.

Soundness and completeness

Soundness

We want to use linear hypergraphs as a graphical representation of our term language.

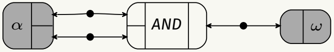
Definition (Soundness)

The language of linear hypergraphs is sound if any terms that are equal in the language of terms have isomorphic interpretations as linear hypergraphs.

Interpreting terms as graphs

We translate terms into linear hypergraphs with the interpretation $[\![-]\!] : \textbf{Term} \to \textbf{Graph}$

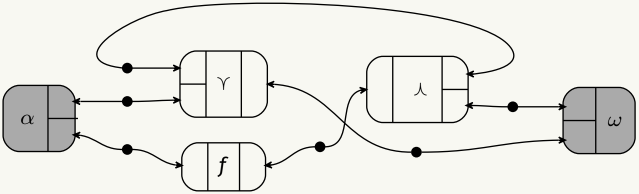
For a gate $\boldsymbol{k}$, $[\![\boldsymbol{k}]\!]$ is defined as a hyperedge with the appropriate number of sources and targets.



We then combine these hyperedges as with our term language to form larger hypergraphs.

Interpreting terms as graphs

$$[\![\mathrm{Tr}^1(\curlyvee \otimes f \cdot \times_{1,1} \cdot \curlywedge \otimes 1)]\!]$$



Soundness

To ensure soundness all the structural axioms must still hold in the language of hypergraphs.

Fortunately they do!

Theorem (Soundness)

For any terms F and G, if $F = G$ by the structural axioms, then their interpretations as linear hypergraphs are isomorphic $[\![F]\!] \equiv [\![G]\!]$.

Completeness

We also need to recover terms from hypergraphs.

Definition (Completeness)

Hypergraphs are a complete graphical language if for any linear hypergraph H there exists a unique term F, up to the structural axioms, such that $[\![F]\!] \equiv H$.

There are two steps to this: definability and coherence.

Definability

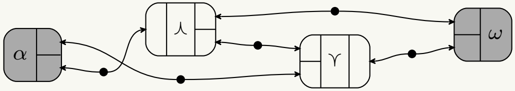
Definition (Definability)

Hypergraphs are definable if for every well-formed hypergraph H we can retrieve a well-formed term for which the hypergraph interpretation of that term is equivalent to the original graph, i.e.

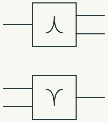
$$[\![\mathrm{term}(H)]\!] \equiv H.$$

Stacking

First we set an order $\leq$ on our edges and stack them up.

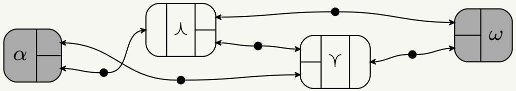


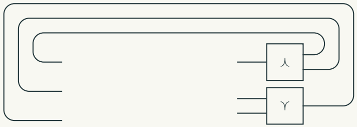
$$\curlywedge \otimes \curlyvee$$



Tracing

Then we trace around all the outputs of the stack:

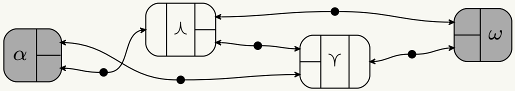


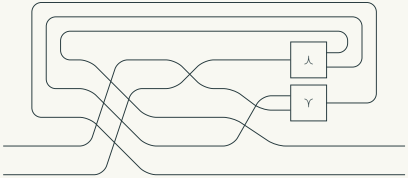


$\mathrm{Tr}^3(? \cdot \curlywedge \otimes \curlyvee)$

Shuffling

We introduce the input wires, and then shuffle everything so all the wires connect to the right place.





$\mathrm{Tr}^3(\times_{3,2} \cdot \times_{1,1} \otimes \mathbf{3} \cdot \mathbf{1} \otimes \times_{2,1} \otimes \mathbf{1} \cdot \curlywedge \otimes \curlyvee \otimes \mathbf{2})$

(**Exercise:** follow around the wires and make sure this is true)

Definability

Through this construction we can always retrieve a term from a linear hypergraph.

Proposition (Definability)

For every well-formed hypergraph H then $[\![\mathrm{term}_{\leq}(H)]\!] \equiv H$.

Whatever edge order we choose at the beginning, we always get a term equal by the axioms!

Proposition (Coherence)

For all orderings of edges $\leq_x$ on a hypergraph F,

$$\mathrm{term}_{\leq_1}(H) = \mathrm{term}_{\leq_2}(H) = \cdots = \mathrm{term}_{\leq_n}(H)$$

Completeness

Theorem (Completeness)

For any linear hypergraph H there exists a unique term F, up to the structural axioms, such that $[\![F]\!] \equiv H$.

This means we can reason purely graphically, and can easily identify any redexes.

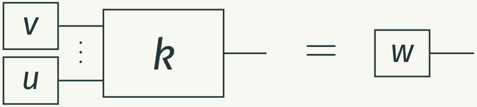
Semantics of digital circuits

Combinational circuits

Gate

Gates are extensional and monotonic.

$$v \otimes \cdots \otimes u \cdot k = w$$

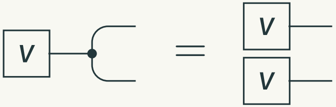


For example, $\mathsf{t} \otimes \mathsf{t} \cdot AND = \mathsf{t}$.

Combinational circuits

Fork

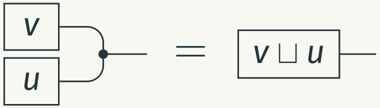
$$V \cdot \curlywedge = V \otimes V$$



Combinational circuits

Join

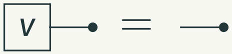
$$v \otimes u \cdot \curlyvee = v \sqcup u$$



Combinational circuits

Stub

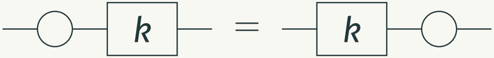
$$V \cdot \sim = \sim$$



Delay

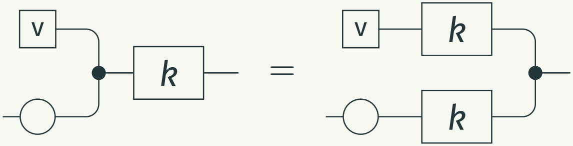
Timelessness

$$\delta \cdot k = k \cdot \delta$$



Streaming

$$v \otimes \delta \cdot \curlyvee \cdot k = (v \cdot k) \otimes (\delta \cdot k) \cdot \curlyvee$$

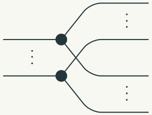


Product

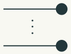
Feedback and recursion are slightly more complex.

First we generalise our fork and stub combinators to act on buses of arbitrary width.

$$\Delta_n : n \to n + n$$



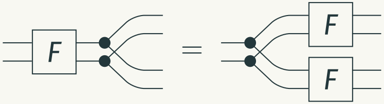
$$\diamond_n : n \to 0$$



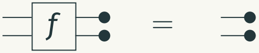
Product

Naturality axioms

$$F \cdot \Delta_n = \Delta_n \cdot F \otimes F$$



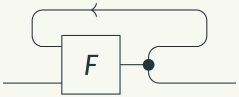
$$F \cdot \diamond_n = \diamond_n$$



Iterator

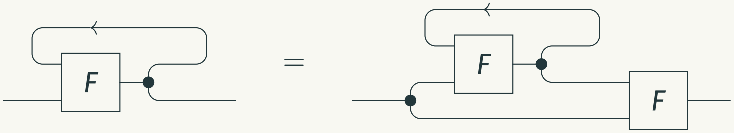
With trace and product we can use the iterator, which lets us model recursion.

$$\text{iter}(F) = \text{Tr}^n(F \cdot \Delta_n)$$



By unfolding the iterator we can model recursion.

$$\text{iter}(F) = \langle \text{iter}(f), m \rangle \cdot F$$



Reducing circuits

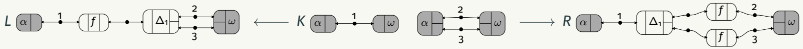
So how do we apply these redexes?

We use graph rewrites.

Graph rewriting

Rewrite rules

A rewrite rule $L \Rightarrow R$ in the language of linear hypergraphs is a span of hypergraph homomorphisms $L \leftarrow K \rightarrow R$, where K is the 'common interface' of L and R.



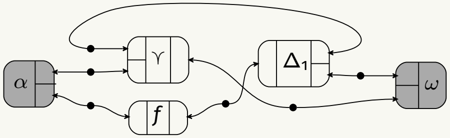
For a set of axioms $\mathcal{E}$ we write $[\![\mathcal{E}]\!]$ for their conversion into the hypergraph language.

DPO rewriting

A common framework for graph rewriting is known as double pushout (DPO) rewriting.

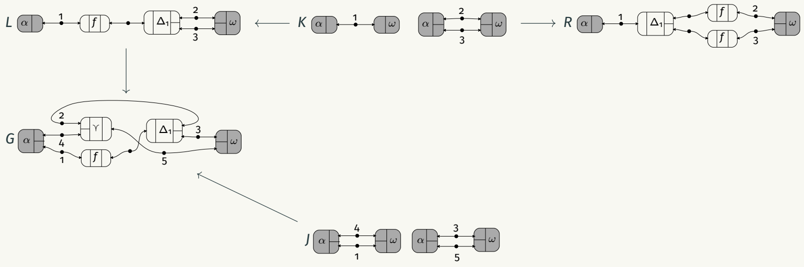
Let's apply it to the term below, which contains one of the product axioms.

$$\mathrm{Tr}^1(\curlyvee \otimes f \cdot \times_{1,1} \cdot \Delta_1 \otimes 1)$$



DPO rewriting

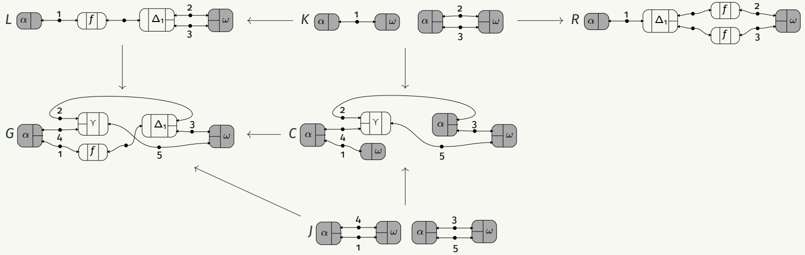
To perform DPO rewriting on a linear hypergraph G with ‘interface’ J, we must first identify a matching morphism $L \to G$.



DPO rewriting

We then compute the pushout complement $K \to C \to G$.

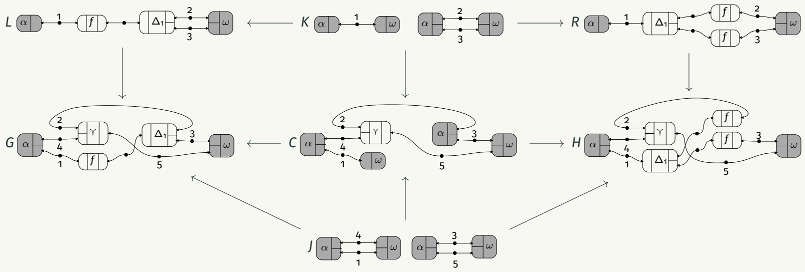
The hypergraph C is effectively 'G with L cut out'.



DPO rewriting

Then we perform a pushout on $C \leftarrow K \rightarrow R$ to obtain our final hypergraph H.

This is 'G with the subgraph L replaced with R'.



We write $G \rightsquigarrow_{[\![\mathcal{E}]\!]} H$ if rewriting can be performed in this way.

Adhesive categories

Not all structures are compatible with DPO rewriting – most importantly, the pushout complement must be unique.

The framework of adhesive categories is often used to ensure that pushout complements are always unique, if they exist.

Theorem

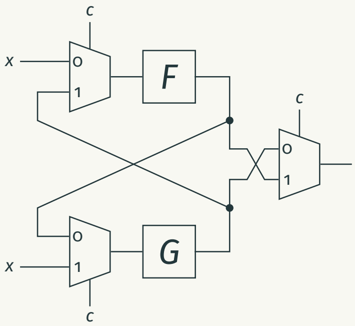
The language of linear hypergraphs is a partial adhesive category.

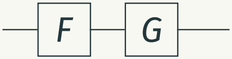
This means for that the pushout complement is unique for a certain class of spans.

This includes our rewrite rules!

Applications

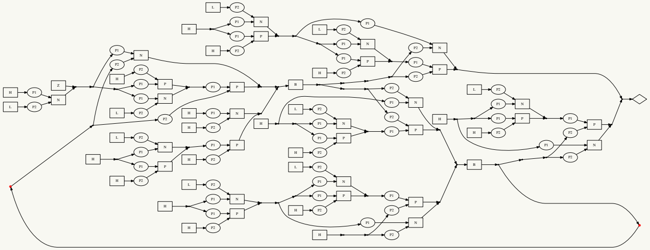
Cyclic combinational circuits

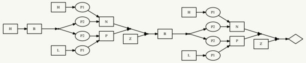




Applications

Pre-logical circuits





Conclusion

We have defined a combinatorial graphical language for symmetric traced monoidal categories, and shown that it is sound and complete.

This allows us to reason about circuits purely graphically.

Even when adding behavioural axioms to our framework, we can still reason graphically with graph rewrites, since linear hypergraphs form a partial adhesive category.

To reason about circuits diagrammatically, we simply formulate the axioms as rewrite rules in the language of hypergraphs.

Do we have all the axioms?

References i

F. Bonchi, F. Gadducci, A. Kissinger, P. Sobociński, and F. Zanasi.
Rewriting modulo symmetric monoidal structure.
In *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10. IEEE, 2016.

H. Ehrig, M. Pfender, and H. J. Schneider.
Graph-grammars: An algebraic approach.
In *14th Annual Symposium on Switching and Automata Theory (swat 1973)*, pages 167–180. IEEE, 1973.

References ii

M. Hasegawa.
On traced monoidal closed categories.
Mathematical Structures in Computer Science, 19(2):217–244, 2009.

A. Kissinger.
Pictures of processes: Automated graph rewriting for monoidal categories and applications to quantum computing, 2012.

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