Mapping Automata to zk-SNARKS Using Category Theory

F. Genovese, A.Knispel, J. Fitzgerald
Statebox Team
research@statebox.io

We provide a categorical procedure to turn graphs corresponding to automata state spaces into boolean circuits, leveraging on the fact that boolean circuits can be easily turned into zk-SNARKS. Our circuits verify that a given sequence of edges and nodes is indeed a path in the graph they represent. We then generalize to circuits verifying paths in arbitrary graphs. We prove that all of our correspondences are pseudofunctorial, and behave nicely with respect to each other.

1 Introduction

Lately, especially due to the advent of smart contracts in business applications, there has been a renewed interest towards classical results in theoretical computer science.

Smart contracts, especially if hosted on blockchain [], are immutable pieces of code, more often than not used to manage money. These are very good reasons to look into ways of writing smart contracts that are reliable, easy to analyze and correct-by-construction. These requirements renewed interest in formal models of computation []: In particular, automata [] are considered easy to implement, very structured, and make possible to prove roperties of the computations being performed.

On the other hand, blockchain also spawned a renewed interest for cyptography, both for security, privacy, and space reasons: It is paramount for blockchains to be cryptographically secure, if they are meant to work as exchanges of valuables of any sort (such as digital currency). As for privacy, criptographic tools such as *zero-knowledge succinct non-interactive argument of knowledge* (zk-SNARKS) [] allow for the verification that an information is correct without actually revealing nothing about the information itself. This has been used, for instance, by ZCash [] to implement private blockchain transactions: Here, transacting parties submit zk-SNARKs of their transactions to the blockchain, and miners then verify that a given transaction followed the rules of the protocol by verifying the zk-SNARK, without gaining any information about who-sent-money-to-who, and how much. Regarding space, it has to be noted that by design blockchains tend to grow indefinitely space-wise as new blocks keep being added to the chain. This is a serious issue, since new nodes are either forced to download many gigabytes of data to sync with the network or they have to require the current state of the chain from another node (which requires trusting the node) and then start syncing from there. This "trust Vs. feasibility" issue can be resolved by recursive zk-SNARKS [], in which snarks are used to verify, using a few Kbs, that the current state received by a node is valid. Such applications look very promising especially in contexts such as blockchain applied to the Internet of Things [].

In this work, we put together formal models of computation and cryptography, providing a categorical way to turn automata into zk-SNARKs that verify how a sequence of inputs leading to a state change follows the rules specified by the automaton itself. To do this, we bypass the problem of modelling cryptographical primitives categorically, using the fact that boolean circuits can be easily turned into zk-SNARKS by already available techniques.

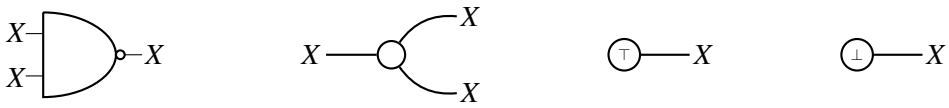
Submitted to:
Working paper

We proceed as follows: In Section 2 we define boolean circuits from a categorical perspective. In Section 3 we briefly explain the links between automata and free categories. In Section 4 we show how to turn a given sequence of state changes for a given automaton into a boolean circuit. We then obtain a generalized boolean circuit which verifies arbitrary sequences up to a given length, and show how it can be turned into a zk-SNARK. In Section 5 we generalize to circuits which accept the specification defining an automaton as input, thus attaining full privacy. In Section 6 we conclude by defining directions of future work.

2 The categories $\mathbb{B}_{\mathbf{fun}}$, $\mathbb{B}_{\mathbf{circ}}$, $\mathbb{B}_{\mathbf{KP}}$

Definition 2.1. *A* boolean function *is a function $\mathbb{B}^n \to \mathbb{B}$, for some natural n. A* generalized boolean function *is a function $\mathbb{B}^n \to \mathbb{B}^m$, for naturals m,n. We denote with $\mathbb{B}_{\boldsymbol{fun}}$* the category of generalized boolean functions*, having $\mathbb{B}^n$, for each natural n as objects, and generalized boolean functions as morphisms. Composition is the usual function composition. This category is clearly symmetric monoidal, with $\mathbb{B}^0$ as unit, and the usual product of functions as product.*

Definition 2.2. *A* boolean circuit *is a wiring of logical gates that computes a boolean function $\mathbb{B}^n \to \mathbb{B}$. A* generalized boolean circuit *is a wiring of logical gates that computes a generalized boolean function $\mathbb{B}^n \to \mathbb{B}^m$. We denote with $\mathbb{B}_{\boldsymbol{circ}}$* the category of generalized boolean circuits*, that is, the free symmetric strict monoidal category generated by one object, denoted with X, and the following four morphisms:*



Respectively called `NAND`, `COPY`, `TRUE`, `FALSE`. We will often use X^n to denote the n-fold monoidal product of X, and X^0 to denote the monoidal unit. For more information about how to generate a free symmetric strict monoidal category from a set of object and morphism generators, see [].

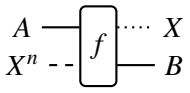
As the name suggests, we interpret  as the `NAND` gate,  as the gate that copies a bit, and `TRUE` and `FALSE` as constants. We can reify this interpretation using the following lemma:

Lemma 2.3. *There is a strict monoidal functor $\texttt{ext} : \mathbb{B}_{\boldsymbol{circ}} \to \mathbb{B}_{\boldsymbol{fun}}$ sending I to $\mathbb{B}^0$, X to $\mathbb{B}$,* *to the usual `NAND` function $\mathbb{B}^2 \to \mathbb{B}$,* *to the cartesian diagonal on $\mathbb{B}$, and $\top, \bot$ to the functions $\mathbb{B}^0 \to \mathbb{B}$ corresponding to the constants* 1 *and* 0*, respectively.*

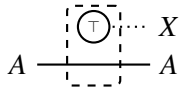
Indeed, there are multiple choices for the generators of $\mathbb{B}_{\mathbf{circ}}$. We decided to use `NAND` because it is well known that this gate, along with the possibility to copy bits, is enough to generate any n-ary logical circuit []. In the following, we will often refer to other gates, such as  (called *`OR`*) or  (*`AND`*): In our setting, these are just syntactic sugar for the opportune wirings of and to simulate them.

Definition 2.4. *We denote with $\mathbb{B}_{\boldsymbol{KP}}$ the* bicategory of knowledge proof circuits *defined as follows:*

- Obj $\mathbb{B}_{\boldsymbol{KP}} :=$ Obj $\mathbb{B}_{\boldsymbol{circ}}$*;*
- Mor $\mathbb{B}_{\boldsymbol{KP}}(A,B) :=$ Mor $\mathbb{B}_{\boldsymbol{circ}}(A \otimes X^n, X \otimes B)$*, for all $n \in \mathbb{N}$. We depict morphisms as shown below; the X^n and X wires are "silent" with respect to our categorical structure, so we depict them dashed and dotted, respectively:*

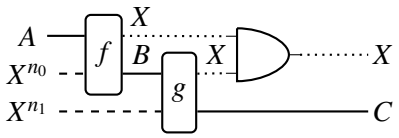


- $id_A := \top \otimes id_A : A \otimes X^0 \to X \otimes A$. *Identites are depicted as follows:*



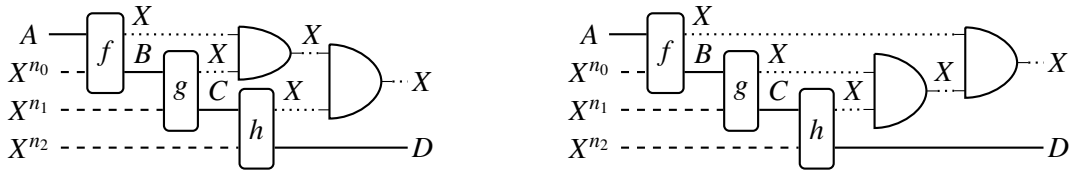
- $\mathrm{Mor}\,\mathbb{B}_{\boldsymbol{KP}}(A,B)(f,g) = \begin{cases} \{*\} & \textit{iff } \mathtt{ext}(f) = \mathtt{ext}(g); \\ \emptyset & \textit{otherwise} \end{cases}$

- *Given $f : A \to B$ and $g : B \to C$, corresponding to morphisms of $\mathbb{B}_{\boldsymbol{circ}}$ $A \otimes X^{n_0} \to X \otimes B$ and $B \otimes X^{n_1} \to X \otimes C$, respectively, we set $f;g$ to be the morphism $(f \otimes id_{X^{n_1}});(id_X \otimes g);(\sqcap \otimes id_C)$. Composition is depicted graphically as follows:*



- *The 2-cell compositions and identities are trivial, and defined in the obvious way.*

The reason why we define $\mathbb{B}_{\mathbf{KP}}$ as a bicategory is that our 1-cell composition in $\mathbb{B}_{\mathbf{KP}}$ is not associative. Indeed, $(f;g);h$ and $f;(g;h)$ are different morphisms, as one can see in the figure below:



The point though is that these morphisms implement the same boolean function, and are extensionally equal: In fact, it is not difficult to check that $\mathtt{AND}(\mathtt{AND}(x,y),z) = \mathtt{AND}(x,\mathtt{AND}(y,z))$ for each triplet of bits x,y,z. A similar argument can be made for identity laws, noting that $\mathtt{AND}(x,1) = x = \mathtt{AND}(1,x)$ for each bit A. For these reasons we introduced 2-cells when $\mathtt{f} = \mathtt{g}$, which capture exactly the notion of extensional equality. Such cells are by construction invertible and give a very trivial 2-structure, where every 2-homset is both a preorder and a groupoid, and bicategory axioms hold on the nose. A more refined definition where 2-cells are circuit rewritings could have been given, but we are not interested in studying circuit rewriting in this work, so we opted for the easiest solution.

3 Finite State Machines (FSMs)

We see *state machines* as Petri nets where each transition has only one inbound and one outbound arc, and all markings have exactly one token. In this setting, while the usual underlying structure of a Petri net is an hypegraph, the underlying structure of a state machine is just a graph. Another way to put this is that we are freely confusing state machines with their state spaces.

Definition 3.1. *A* finite state machine *is a state machine whose underlying graph has a finite number of vertexes and edges.*

We can use a Petri net to generate a free symmetric strict monoidal category, essentially using its underlying hypegraph structure to define object and morphism generators []. In the case of FSMs, the restriction of their underlying hyergraphs to be graphs simplifies things:

Definition 3.2. *To each FSM M we can assign a* category of executions of *M, denoted $\mathfrak{F}(M)$, which is just the free category built on the underlying graph of M. More in detail, the objects of $\mathfrak{F}(M)$ are the vertexes of the underlying graph of M (its vertexes), while morphisms are generated by freely composing the edges of the graph. Identities are the null paths. $\mathfrak{F}(-)$ is a functor **Graph** $\to$ **Cat**. It also has a right adjoint, denoted $\mathfrak{U}(-)$.*

Given a FSM M, every morphism in $\mathfrak{F}(M)$ represents a possible run of M. The goal for the next section will be to functorially map executions into generalized boolean circuits. Then, we will have to turn these generalized circuits into boolean circuits, which verify that a given execution is correct – meaning that all the actions performed correspond to a valid path on the graph.

4 Turning executions into circuits

The first thing to note is that since our graphs are finite, we can enumerate their edges and vertexes. We are designing circuits, so is important to understand how many bits we need for the enumeration. This is seen to be $\lceil \log_2 n \rceil$, where n is the number of elements we need to enumerate. This poses another problem: Suppose we have a graph with, say, 6 vertexes. We will need at least 3 bits to enumerate them. Since $2^3 = 8$, we will have two numbers not corresponding to any vertex in our enumeration. How do we distinguish between numbers enumerating elements and numbers that do not? We propose the following solution: First, for each graph G with vertexes V and edges E, we define functions $V \to 2^{\lceil \log_2(|V|+1) \rceil}$ and $E \to 2^{\lceil \log_2(|E+V|) \rceil}$, such that no vertex is mapped to $0 \dots 0$ – the first number of the enumeration, from now on also denoted as $\mathbf{0}$ – and no edge is mapped to the first V numbers of the enumeration.

The point is that $\mathbf{0}$ is reserved in vertex enumerations, and is meant to signify `undefined`. The first $|V|$ numbers in the edge enumeration are instead reserved to represent the identity morphisms on each vertex in $\mathfrak{F}(G)$.

Having enumerated vertexes and edges, from the structure of the graph we can obtain two tables with the following structure template, respectively called *source* and *target table*:

	id_{v_1}	$\cdots$	id_{v_n}	e_1	$\cdots$	e_m	u_1	$\cdots$	u_k
$\mathbf{0}$	0	$\cdots$	0	0	$\cdots$	0	0	$\cdots$	0
v_1	1	$\cdots$	0	?	$\cdots$	?	0	$\cdots$	0
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
v_n	0	$\cdots$	1	?	$\cdots$	?	0	$\cdots$	0
u'_1	0	$\cdots$	0	0	$\cdots$	0	0	$\cdots$	0
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
u'_h	0	$\cdots$	0	0	$\cdots$	0	0	$\cdots$	0

Here we have that $n+h+1 = 2^{\lceil \log_2(|V|+1) \rceil}$, and $n+m+k = 2^{\lceil \log_2(|E+V|) \rceil}$. The v_i are the enumerations of the vertexes, the e_i enumerations of the edges, and the u_i, u'_i represent the unassigned edge and vertexe enumerations, respectively. In the source (resp. target) table, we put a 1 in a position if a given vertex is the source (resp. target) of a given morphism. Since the first n enumerations for the vertexes are reserved for identity morphisms, this forces our choices in the first n columns. Similarly, since the u_i and u'_i are undefined, there are 0s in all the entries indexed by them. The question marks represent the fact that there may be a 0 or a 1 in that position, as long as there is just one 1 in each of those columns (an edge can only have one vertex).

4.1 Basic circuits

Using our tables, we are able to build a couple of generalized boolean functions, where we denoted with $\mathbb{B}^V$ and $\mathbb{B}^E$ the sets $\mathbb{B}^{\lceil \log_2(|V|+1)\rceil}$ and $\mathbb{B}^{\lceil \log_2(|E+V|)\rceil}$ respectively:

$$\mathrm{s}_G(\_), \mathrm{t}_G(\_) : \mathbb{B}^E \to \mathbb{B}^V$$

These functions take in input the enumeration of an edge, and return the enumeration of its source and target vertex, respectively. If the input corresponds to an undefined edge, then they return **0**.

The next step is to consider a "matching function" $\supset\!\!\bullet\!\!\cdots_n : \mathbb{B}^n \otimes \mathbb{B}^n \to \mathbb{B}$, for each n, which has the following behaviour:

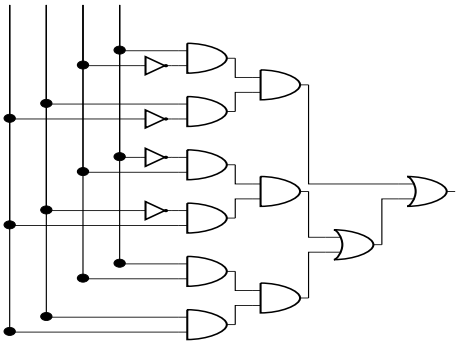
$$\supset\!\!\bullet\!\!\cdots_n(x,y) := \begin{cases} 1 \text{ iff } (x = y) \wedge (x, y \neq \mathbf{0}); \\ 0 \text{ otherwise.} \end{cases}$$

Essentially, $\supset\!\!\bullet\!\!\cdots_n$ matches inputs but returns 0 if one of the inputs is undefined.

The fact that we can build $\mathrm{s}_G(\_)$, $\mathrm{t}_G(\_)$ and $\supset\!\!\bullet\!\!\cdots_n(\_,\_)$ follows trivially from the fact that the function space $\mathbb{B}^E \to \mathbb{B}^V$ is finite. We conclude by making use of the following theorem, with which we can assume that there are generalized boolean circuits S_G, T_G and $\supset\!\!\bullet\!\!\cdots_{X^V}$ implementing $\mathrm{s}_G(\_)$, $\mathrm{t}_G(\_)$ and $\supset\!\!\bullet\!\!\cdots_{X^V}(\_,\_)$, respectively.

Theorem 4.1 (reference)**.** *For every generalized boolean function, there is a generalized boolean circuit that evaluates it.*

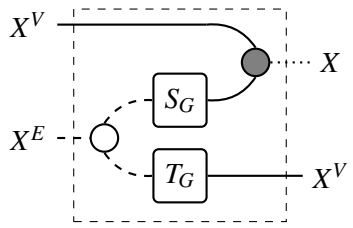
An example of a circuit implementing $\supset\!\!\bullet\!\!\cdots_2$ (so for 2 bits) together with its truth table is the following:



a	b	$\supset\!\!\bullet\!\!\cdots_2(a,b)$
0	**0**	0
01	**0**	0
10	**0**	0
11	**0**	0
0	01	0
01	01	1
10	01	0
11	01	0
0	10	0
01	10	0
10	10	1
11	10	0
0	11	0
01	11	0
10	11	0
11	11	1

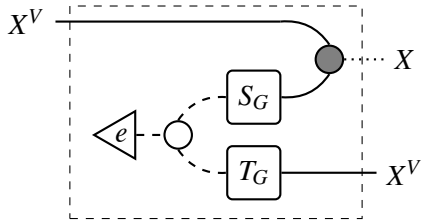
4.2 Mapping paths

Denoting with $-\!\!\subset_{X^E}$ the `COPY` circuit acting on E bits, we now notice that the generalized boolean circuit $(id_{X^V} \otimes -\!\!\subset_{X^E}); (id_{X^V} \otimes S_G \otimes T_G); (\supset\!\!\bullet\!\!\cdots_{X^V} \otimes id_{X^V})$, when mapped to $\mathbb{B}_{\mathbf{fun}}$ through `ext`, will correspond to the function accepting a vertex and an edge enumeration in input, and will return 1 if the vertex is the source of the edge, 0 otherwise, along with the enumeration of the edge's target. Importantly, it will always return 0 on the first output for any undefined enumeration in input. It is, moreover, a morphism in $\mathbb{B}_{\mathbf{KP}}$, as it becomes evident by drawing it:



Theorem 4.2. *Having chosen an enumeration on the vertexes and edges of a graph G, there is a pseudofunctor $\mathfrak{F}(G) \to \mathbb{B}_{\boldsymbol{KP}}$, sending each object to X^V, and each generating morphism e of $\mathfrak{F}(G)$ to the following morphism, called e-*evaluator*, where e represents the constant gate outputting the enumeration of e when considered as an edge of G:*

$$(id_{X^V} \otimes e);(id_{X^V} \otimes \multimap_{X^E});(id_{X^V} \otimes S_G \otimes T_G);(\supset_{X^V} \otimes id_{X^V})$$



The image of $\mathfrak{F}(G)$ through this pseudofunctor is called $\mathbb{B}^G_{\boldsymbol{path}}$, the category of path proofs over *G*.

The circuits of Theorem 4.2 have the disadvantage of working on fixed paths, while we would like a general circuit working with every path of a given graph. To solve this problem, we take an intermediate step:

Lemma 4.3. *Consider the category **Count**, which has one object $*$ and natural numbers as morphisms, with 0 is the identity morphism and composition as addition.*

*For each graph G, there is a functor $\mathfrak{F}(G) \to$ **Count** sending every object to $*$, identities to 0, and generating morphisms to 1. This extends to a functorial correspondence between **Graph** and the category of endofunctors over **Count**.*

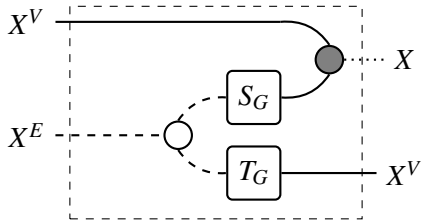
Count is a category that, as the name suggests, counts how many generating morphisms compose a path. We can use it to shape general circuits that work for every path in a graph.

Theorem 4.4. *For a graph G, consider an enumeration and S_G and T_G as defined in Theorem 4.2. There is a pseudofunctor **Count** $\to \mathbb{B}_{\boldsymbol{KP}}$ sending $*$ to X^V, 0 to id_{X^V} and $n > 0$ to the n-fold composition of the morphism*

$$(id_{X^V} \otimes \multimap_{X^E});(id_{X^V} \otimes S_G \otimes T_G);(\supset_{X^V} \otimes id_{X^V})$$

*The composition of this pseudofunctor with the functor of Lemma 4.3 gives a pseudofunctor $\mathfrak{F}(G) \to$ **Count** $\to \mathbb{B}_{\boldsymbol{KP}}$ sending each object to X^V, and each generating morphism to the circuit:*

$$(id_{X^V} \otimes \multimap_{X^E});(id_{X^V} \otimes S_G \otimes T_G);(\supset_{X^V} \otimes id_{X^V})$$



The image of $\mathfrak{F}(G)$ through this pseudofunctor is called $\mathbb{B}^G_{\boldsymbol{Graph}}$, the category of proofs over *G*.

The pseudofunctor in Theorem 4.4 associates to each path of length m, seen as a morphism in $\mathfrak{F}(G)$, a generalized boolean circuit. This circuit accepts the enumeration of a vertex v and a path of n edges

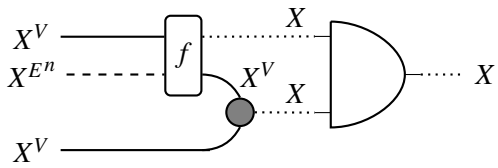
(specified as n enumerations of edges) as inputs, and returns 1 and an enumeration of v' in output if the path leads from v to v'. It returns 0 and $\mathbf{0}$ otherwise. Notice that since we included identities in the truth tables when defining S_G, T_G, we are also able to verify *any path of length less than n by padding any path with identities*.

4.3 Snarkizing circuits

How do we turn the morphisms in $\mathbb{B}^G_{\mathbf{Graph}}$ into zk-SNARKS? Luckily enough, it turns out we do not have to build zk-SNARKS ourselves. Indeed, there are already implemented ways to turn boolean circuits into zk-SNARKS []. Figuring out a cryptographically secure way to turn circuits into zk-SNARKS is no simple endeavour, that would probably take years and extensive security auditing. Instead, we deem a wiser course of action turning generalized boolean circuits in $\mathbb{B}^G_{\mathbf{Graph}}$ into boolean circuits, and feed them to an already implemented and audited solution.

Definition 4.5. *For each graph G we define the* snarkizator *as a function* $\mathrm{Sn}(\_) : \mathrm{Mor}\, \mathbb{B}^G_{\boldsymbol{Graph}} \to \mathrm{Mor}\, \mathbb{B}_{\boldsymbol{circ}}$ *that maps a morphism* $f : X^V \to X^V$ *to*

$$(f \otimes id_{X^V}); (id_X \otimes \supset\!\bullet\!\cdots_{X^V}); \supset$$



Notice how a snarkized circuit just outputs a bit, and is thus a boolean circuit. Note moreover how the function $\mathrm{Sn}(\_)$ cannot be made improved to a (pseudo)functor, since boolean circuits themselves do not form a category.

For each morphism f, $\mathrm{Sn}(f)$ is a boolean circuit which takes two values a, b of type X^V as input, representing vertexes, along with $f_1, \ldots, f_n$ inputs of type X^E, representing edges, and returns 1 if and only if the edge inputs define a valid path from a to b according to the graph specification defined by S_G and T_G. The corresponding zk-SNARK, obtained by simply feeding our circuit to any already available library such as `libsnark` [], is a succint zero knowledge proof that any specified path in the graph is valid or not.

5 Abstracting over graphs

Up to now, circuits in $\mathbb{B}^G_{\mathbf{Graph}}$ have the problem that the topology of G is used to define S_G and T_G, and is thus hardwired in the circuit. Since in creating zk-SNARKs some information has to be necessarily made public [], this may cause problems. Indeed, it may be possible to reverse-engineer this public information to obtain information about G. Albeit this would still allow to keep used paths secret, the topology of the state space of an automaton can reveal a lot about what the automaton is used for. We would like to keep this information private.

To do this, we notice that as S_G and T_G are obtained by building a truth table from the adjacencies of G, this truth table could be fed itself to a "universal function" that builds S and T for all graphs up to a given size. In detail, if n, m are numbers, we can consider boolean functions

$$\mathrm{s}_{m,n}(\_), \mathrm{t}_{m,n}(\_) : \mathbb{B}^{f(m,n)} \times \mathbb{B}^m \to \mathbb{B}^n$$

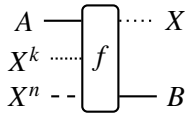
Which take m bits in input, representing an enumeration of the edges of a graph whose source and target matrices are specified by $f(m,n)$ bits, and return n bits, representing an enumeration of their source and target, respectively. Notice how we write $f(m,n)$ since the size of the adjacency matrixes defining a graph will in general depend on the maximum number of vertexes and edges we are allowing. As before, we can invoke Theorem 4.1 to consider implementations $S_{m,n}$ and $T_{m,n}$ of $\mathrm{s}_{m,n}(\_)$ and $\mathrm{t}_{m,n}(\_)$, respectively.

Since we have introduced new inputs, just substituting $S_{m,n}$ and $T_{m,n}$ in place of S_G and T_G in Theorems 4.2 and 4.4 won't work: Indeed, it would force us to specify the encoding for G n times in a n-fold composition. Moreover, what happens if we given different graphs specifications in the composition?

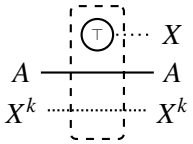
The point is that notice that the categorical structure of $\mathbb{B}_{\mathbf{KP}}$ is not good to manage inputs that have to be routinely repeated. We fix these problems straight away by defining a new category as follows:

Definition 5.1. *Fix a natural number k. We denote with $\mathbb{B}^k_{\mathbf{ZKP}}$ the* bicategory of zero knowledge proof circuits of size *k defined as follows:*

- $\mathrm{Obj}\ \mathbb{B}^k_{\mathbf{ZKP}} := \mathrm{Obj}\ \mathbb{B}_{\mathbf{circ}}$;
- $\mathrm{Mor}\ \mathbb{B}^k_{\mathbf{ZKP}}(A,B) := \mathrm{Mor}\ \mathbb{B}_{\mathbf{circ}}(A \otimes X^k \otimes X^n, X \otimes B)$, *for all* $n \in \mathbb{N}$. *We depict morphisms as shown below; the X^k, X^n and X wires are "silent" with respect to our categorical structure, so we depict them densely dotted, dashed and dotted, respectively:*



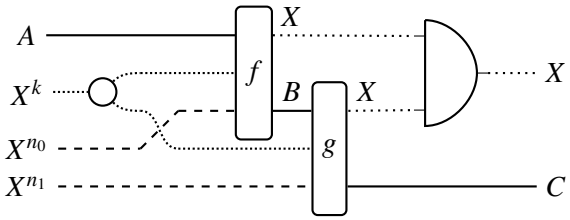
- $id_A := \top \otimes id_A \otimes id_{X^k} : A \otimes X^k \otimes X^0 \to X \otimes A$. *Identites are depicted as follows:*



- $\mathrm{Mor}\ \mathbb{B}^1_{\mathbf{ZKP}}(A,B)(f,g) = \begin{cases} \{*\} & \text{iff } \mathtt{ext}(f) = \mathtt{ext}(g); \\ \emptyset & \text{otherwise} \end{cases}$
- *Given $f : A \to B$ and $g : B \to C$, corresponding to morphisms of $\mathbb{B}_{\mathbf{circ}}$ $A \otimes X^k \otimes X^{n_0} \to X \otimes B$ and $B \otimes X^k \otimes X^{n_1} \to X \otimes C$, respectively, we set $f;g$ to be the morphism*

$$(id_A \otimes \multimap_{X^k} \otimes id_{X^{n_0} \otimes X^{n_1}}); (id_{A \otimes X^k} \otimes \sigma_{X^{n_0},X^k} \otimes id_{X^{n_1}}); (f \otimes id_{X^k \otimes X^{n_1}}); (id_X \otimes g); (\vdash \otimes id_C)$$

Where we denoted with $\sigma_{X^{n_0},X^{k_1}}$ the usual symmetries. Composition is depicted graphically as follows:

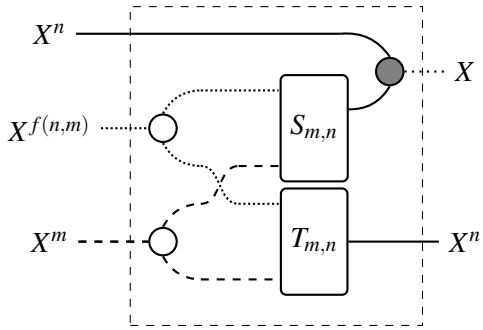


- *The 2-cell compositions and identities are trivial, and defined in the obvious way.*

Proceeding as in Lemma 4.3, we are able to prove the following theorem:

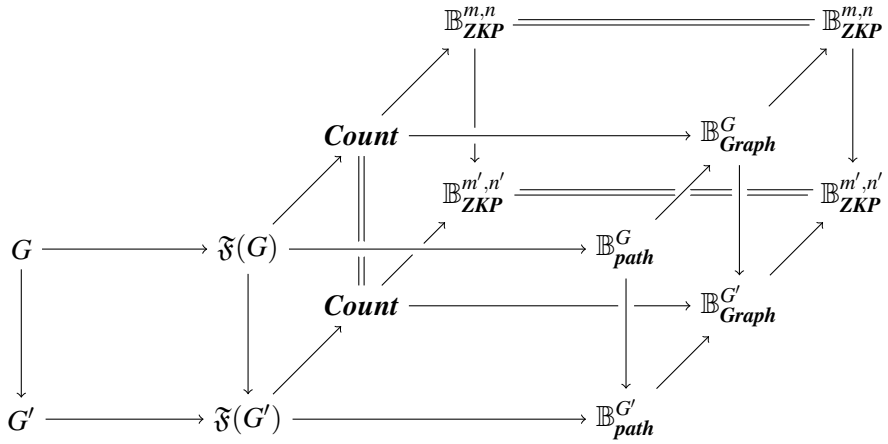
Theorem 5.2. *For each n,m, denote with $f(n,m)$ the function outputting how many bits are needed to store the source and target truth tables for graphs with n vertexes and m edges. There is a functor $\textbf{\textit{Count}} \to \mathbb{B}^{f(n,m)}_{\textbf{\textit{ZKP}}}$ sending each number k to the k-fold composition of the morphism*

$$(id_{X^n} \otimes \multimap_{X^{f(n,m)}} \otimes \multimap_{X^m}); (id_{X^n \otimes X^{f(n,m)}} \otimes \sigma_{X^{f(n,m)},X^m} \otimes id_{X^m}); (id_{X^n} \otimes S_{n,m} \otimes T_{n,m}); (\supset_{X^n} \otimes id_{X^n})$$



We conclude by putting everything together, showing how all the constructions we built behave well with respect to each other.

Theorem 5.3. *Let G, G' be graphs with n,n' vertices and m,m' edges, respectively. Then for each morphism $G \to G'$ and each path p of G the following diagram commutes:*



6 Conclusion and future work

We defined a pseudofunctorial way to turn graphs into families of boolean circuits that can verify the correctness of any path in the graph. Then, we generalized this to circuits that can verify correctness of paths for any graph with a bounded number of vertexes and edges, obtaining a pseudofunctorial correspondence between the category **Graph** and the category of circuits [].

Since graphs can be used to represent finite state machines and boolean circuits can be compiled into zk-SNARKS, this in turn provides a pseudofunctorial way to turn FSMs into zk-SNARKS.

Ongoing work includes implementing our correspondence in a formally verified setting using dependent types. Future work is mainly focused in generalizing our machinery to map free symmetric strict monoidal categories into boolean circuits, providing a way to define arbitrary executions for Petri nets.

Acknowledgements

The authors want to thank the Ehtereum Foundation, that financed this work with a grant.

References

Appendix - Proofs

Theorem 2.3. *There is a strict monoidal functor* $\texttt{ext} : \mathbb{B}_{\textbf{circ}} \to \mathbb{B}_{\textbf{fun}}$ *sending* I *to* $\mathbb{B}^0$, X *to* $\mathbb{B}$, $\supset$ *to the usual* NAND *function* $\mathbb{B}^2 \to \mathbb{B}$, $-\!<$ *to the cartesian diagonal on* $\mathbb{B}$, *and* $\top, \bot$ *to the functions* $\mathbb{B}^0 \to \mathbb{B}$ *corresponding to the constants* 1 *and* 0, *respectively.*

Proof. Strict monoidal functoriality is obvious from the freeness of $\mathbb{B}_{\textbf{circ}}$. □

Lemma .1. $\mathbb{B}_{\textbf{KP}}$ *as defined in Definition 2.4 is a bicategory.*

Proof. First notice that for each A, B, $\mathbb{B}_{\textbf{KP}}(A,B)$ is trivially a category, since function extensionality is reflexive and transitive.

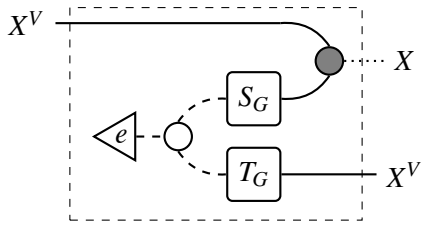
Moreover, composition is clearly functorial. This follows from the fact that extensionality is a congruence with respect to function composition.

The existence of unitors and associators follows from the fact that $\supset(x,1)$ and $\supset(1,x)$ are extensionally equal to the identity function $\mathbb{B} \to \mathbb{B}$, as extensionally equal are $\supset(\supset(\_,\_),\_)$ and $\supset(\_,\supset(\_,\_))$.

Naturality and satisfaction of pentagon and triangle identities for associators and unitors, respectively, follows from the fact that the 2-cell structure of $\mathbb{B}_{\textbf{KP}}$ is posetal, so proving that such morphisms exist is enough. □

Theorem 4.2. *Having chosen an enumeration on the vertexes and edges of a graph* G, *there is a pseudofunctor* $\mathfrak{F}(G) \to \mathbb{B}_{\textbf{KP}}$, *sending each object to* X^V, *and each generating morphism* e *of* $\mathfrak{F}(G)$ *to the following morphism, called* e-evaluator, *where* e *represents the constant gate outputting the enumeration of* e *when considered as an edge of* G*:*

$$(id_{X^V} \otimes e); (id_{X^V} \otimes -\!<_{X^E}); (id_{X^V} \otimes S_G \otimes T_G); (\supset_{X^V} \otimes id_{X^V})$$



The image of $\mathfrak{F}(G)$ *through this pseudofunctor is called* $\mathbb{B}^G_{\textbf{path}}$, *the category of* path proofs over G.

Proof. Obvious from the freeness of $\mathfrak{F}(G)$ and the fact that the bicategorical structure of $\mathbb{B}_{\textbf{KP}}$ is trivial. □

Lemma 4.2. *Consider the category* ***Count****, which has one object* $*$ *and natural numbers as morphisms, with* 0 *is the identity morphism and composition as addition.*

For each graph G, *there is a pseudofunctor* $\mathfrak{F}(G) \to$ ***Count*** *sending every object to* $*$, *identities to* 0, *and generating morphisms to* 1. *This extends to a functorial correspondence between* ***Graph*** *and the category of endofunctors over* ***Count****.*

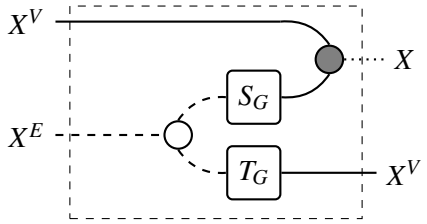
Proof. Functoriality is obvious from the freeness of $\mathfrak{F}(G)$. □

Theorem 4.4. *For a graph G, consider an enumeration and S_G and T_G as defined in Theorem 4.2. There is a pseudofunctor **Count** $\to \mathbb{B}_{\mathbf{KP}}$ sending $*$ to X^V, 0 to id_{X^V} and $n > 0$ to the n-fold composition of the morphism*

$$(id_{X^V} \otimes -\!\!<_{X^E}); (id_{X^V} \otimes S_G \otimes T_G); (\,>\!\!-_{X^V} \otimes id_{X^V})$$

*The composition of this pseudofunctor with the pseudofunctor of Lemma 4.3 gives a pseudofunctor $\mathfrak{F}(G) \to$ **Count** $\to \mathbb{B}_{\mathbf{KP}}$ sending each object to X^V, and each generating morphism to the circuit:*

$$(id_{X^V} \otimes -\!\!<_{X^E}); (id_{X^V} \otimes S_G \otimes T_G); (\,>\!\!-_{X^V} \otimes id_{X^V})$$



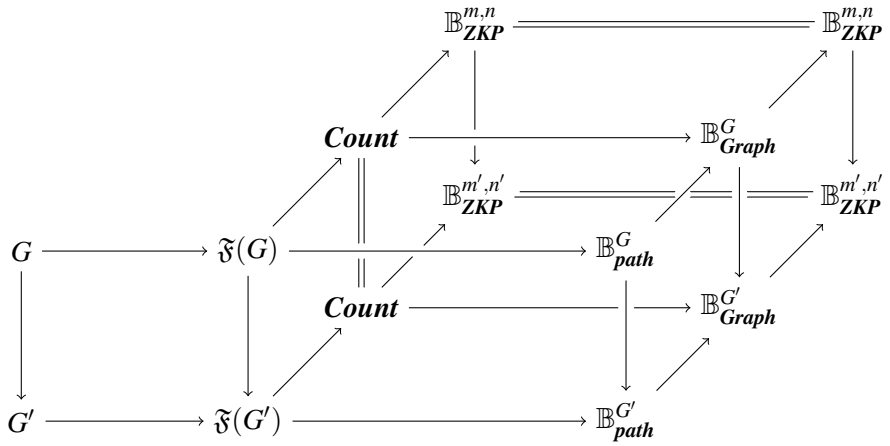
The image of $\mathfrak{F}(G)$ through this pseudofunctor is called $\mathbb{B}^G_{\mathbf{Graph}}$, the category of proofs over *G*.

Proof. The only non-trivial part is proving pseudofunctoriality of the functor **Count** $\to \mathbb{B}_{\mathbf{KP}}$. For this, notice that the same natural number n can be obtained by adding smaller numbers in different orders, and with different bracketings. These will in turn correspond to different ways to compose the same morphism in $\mathbb{B}_{\mathbf{KP}}$, n-times. All these compositions are extensionally equal, which guarantees the existence of isomorphic 2-cells between them. Pseudofunctoriality follows trivially, leveraging on the fact that the 2-cell structure of $\mathbb{B}_{\mathbf{KP}}$ is itself trivial. □

Lemma .2. *For each n, $\mathbb{B}^n_{\mathbf{ZKP}}$ as defined in Definition 5.1 is a bicategory.*

Proof. The proof proceeds exactly as in Lemma .1, by applying function extensionality also to different compositions of $-\!\!<^m_X$ and $\sigma_{X^{f(n,m)},X^m}$. □

Theorem 5.3. *Let G, G′ be graphs with n,n' vertices and m,m' edges, respectively. Then for each morphism $G \to G'$ and each path p of G the following diagram commutes:*



Proof. TODO □