Mapping Finite State Machines 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 state spaces of finite state machines 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 the blockchain [4, 13], 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 [14]: In particular, *finite state machines (FSMs)* [12] are considered easy to implement, well structured, and make possible to prove properties 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)* [1] 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 [8] 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 [11]. 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* [2], which can be used to verify, using just a few Kilobytes, 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 [9].

In this work, we put together formal models of computation and cryptography, providing a categorical way to turn finite state machines into zk-SNARKs that verify how a sequence of inputs leading to a state change follows the rules specified by the finite state machine itself. To do this, we bypass the problem

Preliminary Report. Final version to appear in:
DEVCON V

of modelling cryptographical primitives categorically, using the fact that boolean circuits can be easily turned into zk-SNARKs by already available techniques.

We proceed as follows: In Section 2 we define boolean circuits from a categorical perspective. In Section 3 we briefly explain the links between finite state machines and free categories. In Section 4 we show how to turn a given sequence of state changes for a given finite state machine into a boolean circuit. We then obtain a 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 a finite state machine 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* $\mathbb{B}^n \to \mathbb{B}^m$*, for naturals* m,n*. We denote with* $\mathbb{B}_{\textit{fun}}$ the category of boolean functions*, having* $\mathbb{B}^n$*, for each natural n as objects, and 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.*

We want to give a categorical description of boolean circuits, which are wirings of logical gates that compute a boolean function. The way these circuits are wired is classically modeled by directed acyclic graphs, however we can model them as morphisms in a monoidal category. First, we need to choose a set of gates:

Definition 2.2. *A* set of gates *consists of a family of sets* $G_{n,m}$*, and a family of functions* $\mathtt{int}_{n,m} : G_{n,m} \to (\mathbb{B}^n \to \mathbb{B}^m)$.

Definition 2.3. *Let G be a set of gates. We denote with* $\mathbb{B}^G_{\textit{circ}}$ the category of boolean circuits with gates in *G, which is the free symmetric strict monoidal category generated by one object, denoted X, and morphisms* $m_g : X^n \to X^m$ *corresponding to elements g of* $G_{n,m}$*. 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 [6].*

From there, we get a functor that maps a boolean circuit to the function it computes:

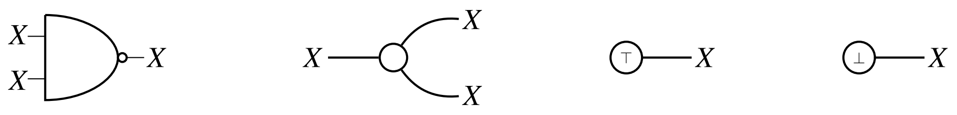
Lemma 2.4. *There exists a strict monoidal functor* $\mathtt{ext}^G : \mathbb{B}^G_{\textit{circ}} \to \mathbb{B}_{\textit{fun}}$ *sending the generating morphisms* m_g *to the function* $\mathtt{int}(g)$.

For our purposes, it is necessary that every boolean function can be computed by a boolean circuit, i.e. that the functor $\mathtt{ext}^G$ is full.

Definition 2.5. *A set of gates is called* functionally complete *if* $\mathtt{ext}^G$ *is full.*

This is a reformulation of the classical definition of functional completeness (see [7]). An important distinction between our formalism an the classic one is that we have to explicitly add the constant gates and a `COPY` gate.

Lemma 2.6. *The set of circuits consisting of* `NAND`*,* `COPY`*,* `TRUE` *and* `FALSE` *is functionally complete. We denote the morphisms they generate as follows:*

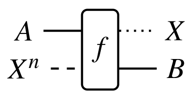


For the remainder of this paper we fix a functionally complete set of circuits and omit the index referring to it. We will refer to specific gates, such as ▷ (`OR`) or ▷ (`AND`): In our setting, these are just syntactic sugar for the opportune circuit that simulates them.

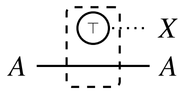
We also need a category that models boolean circuits that allow possibly incorrect inputs as well as extra inputs that are aggregated when morphisms are composed:

Definition 2.7. *We denote with* $\mathbb{B}_{KP}$ *the* bicategory of knowledge proof circuits *defined as follows:*

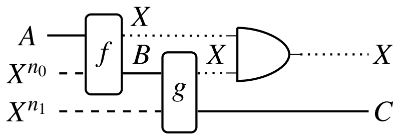
- Obj $\mathbb{B}_{KP} :=$ Obj $\mathbb{B}_{circ}$*;*
- Mor $\mathbb{B}_{KP}(A,B) :=$ Mor $\mathbb{B}_{circ}(A \otimes X^n, X \otimes B)$*, for all* $n \in \mathbb{N}$*. We depict morphisms as shown below; the* X^n *and* X *wires are "silent" with respect to our categorical structure, so we depict them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



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



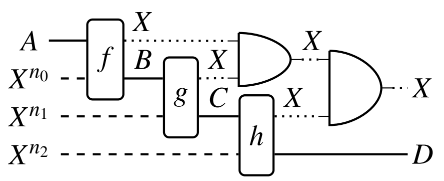
- Mor $\mathbb{B}_{KP}(A,B)(f,g) = \begin{cases} \{*\} & \textit{iff } \mathtt{ext} f = \mathtt{ext} g; \\ \emptyset & \textit{otherwise} \end{cases}$
- *Given* $f : A \to B$ *and* $g : B \to C$*, corresponding to morphisms of* $\mathbb{B}_{circ}$ $A \otimes X^{n_0} \to X \otimes B$ *and* $B \otimes X^{n_1} \to X \otimes C$*, respectively, we set* $f;g$ *to be the morphism* $(f \otimes id_{X^{n_1}});(id_X \otimes g);(▷ \otimes id_C)$*. Composition is depicted graphically as follows:*

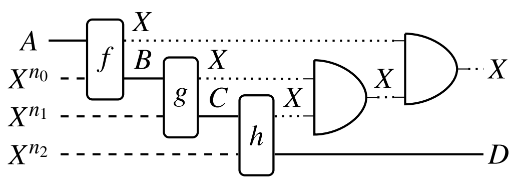


In words, we compose morphisms by wiring the dotted wires together into an `AND` *gate, and by considering the monoidal product of the dashed wires as the dashed wire of the composition.*

- *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 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 $\texttt{ext} f = \texttt{ext} g$, 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 [17, Ch.2] 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 hypergraph, 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 hypergraph structure to define object and morphism generators [6]. 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 [10, pp.49-51]. 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 boolean circuits. Then, we will have to turn these 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*:

$$\begin{array}{c|ccccccccc}
 & id_{v_1} & \cdots & id_{v_n} & e_1 & \cdots & e_m & u_1 & \cdots & u_k \\
\hline
\mathbf{0} & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\
v_1 & 1 & \cdots & 0 & ? & \cdots & ? & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
v_n & 0 & \cdots & 1 & ? & \cdots & ? & 0 & \cdots & 0 \\
u'_1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
u'_h & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0
\end{array}$$

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 vertex 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. We reserve the first n enumerations for the vertexes for identity morphisms; this forces our choices in the first n columns, which along with rows 1 to n define an identity matrix. 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 source/target vertex).

4.1 Basic circuits

Using our tables, we are able to build a couple of 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:

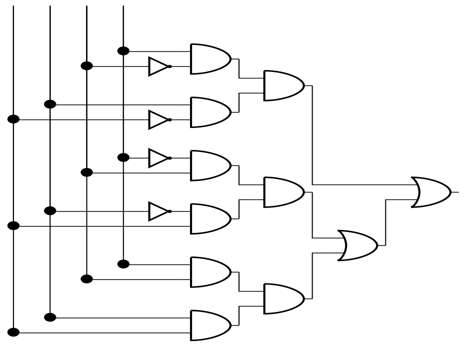
$$\mathrm{s}_G(\_), \mathrm{t}_G(\_) : \mathbb{B}^E \to \mathbb{B}^V$$

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 $\mathbf{0}$.

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

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

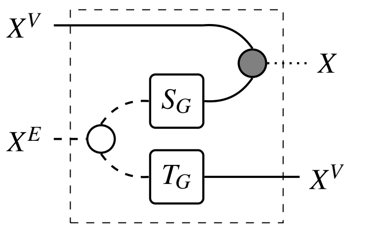
Essentially, $\succ\!\!\bullet_n$ matches inputs but returns 0 if one of the inputs is undefined. We call the boolean circuits implementing $\mathrm{s}_G(\_)$, $\mathrm{t}_G(\_)$ and $\succ\!\!\bullet_{X^V}(\_,\_)$, S_G, T_G and $\succ\!\!\bullet_{X^V}$ respectively. An example of a circuit implementing $\succ\!\!\bullet_2$ (so for 2 bits) together with its truth table is the following:



a	b	$\succ\!\!\bullet_2(a,b)$
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

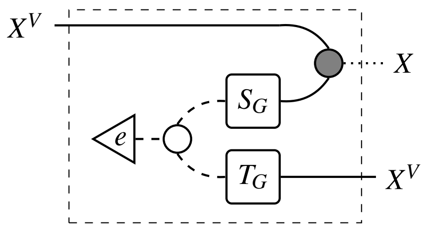
4.2 Mapping paths

Denoting with $-\!\!\prec_{X^E}$ the `COPY` circuit acting on E bits, we now notice that the boolean circuit $(id_{X^V} \otimes -\!\!\prec_{X^E}); (id_{X^V} \otimes S_G \otimes T_G); (\succ\!\!\bullet_{X^V} \otimes id_{X^V})$, 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 becomes evident by drawing it:



Theorem 4.1. *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, where e represents the constant gate outputting the enumeration of e when considered as an edge of G:*

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



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 4.1 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 4.2. *Consider the category **Count**, which has one object $*$ and natural numbers as morphisms, with 0 as 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**.*

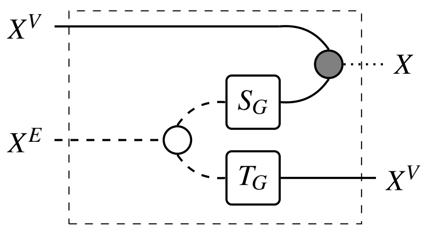
Count is a category that, as the name suggests, counts how many generating morphisms compose a path. We can use it to shape general circuits that work for every path in a graph.

Theorem 4.3. *For a graph G, consider an enumeration and S_G and T_G as defined in Theorem 4.1. 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*

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

*The composition of this pseudofunctor with the functor of Lemma 4.2 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:*

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



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

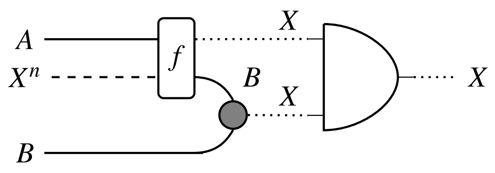
The pseudofunctor in Theorem 4.3 associates to each path of length m, seen as a morphism in $\mathfrak{F}(G)$, a boolean circuit. This circuit accepts the enumeration of a vertex v and a path of n edges (specified as n enumerations of edges) as inputs, and returns 1 and an enumeration of v' in output 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 process *any path of length less than n by padding it 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 [15]. 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 boolean circuits in $\mathbb{B}^G_{\mathbf{Graph}}$ into boolean circuits, and feed them to an already implemented and audited solution.

Definition 4.4. *For each graph G we define the* snarkizator *as a function* $\mathrm{Sn}(\_) : \mathrm{Mor}\,\mathbb{B}_{\textit{KP}} \to \mathrm{Mor}\,\mathbb{B}_{\textit{circ}}$ *that maps a morphism* $f : A \to B$ *to*

$$(f \otimes id_B); (id_X \otimes \rhd\!\!\bullet_B); \rhd$$



Notice how a snarkized circuit just outputs a bit, which is required to turn it into a zk-SNARK. Note moreover how the function $\mathrm{Sn}(\_)$ cannot be improved to a (pseudo)functor, since it does not respect composition.

Since $\mathbb{B}^G_{\mathbf{Graph}}$ is a subcategory of $\mathbb{B}_{\mathbf{KP}}$, for each morphism f in $\mathbb{B}^G_{\mathbf{Graph}}$ we can consider $\mathrm{Sn}(f)$. It 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` [15], is a succint, non-interactive zero knowledge proof that any specified path in the graph – up to length n – 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 [5], 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 a finite state machine can reveal a lot about what the finite state machine 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

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

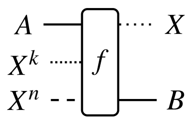
Which take m bits in input, representing an enumeration of the edges of a graph whose source and target matrices are specified by $f(m,n)$ bits, and return n bits, representing an enumeration of their source and target, respectively. Notice how we write $f(m,n)$ since the size of the adjacency matrixes defining a graph will in general depend on the maximum number of vertexes and edges we are allowing. As before, we 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 4.1 and 4.3 won't work: Indeed, it would force us to specify the encoding for G k times in a k-fold composition. Moreover, what happens if we give different graphs specifications in the composition?

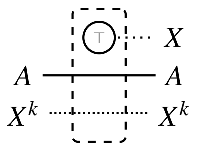
As we see, building on top of Definition 2.7 in this new setting creates possibly pathological outcomes. The point is that the categorical structure of $\mathbb{B}_{\mathbf{KP}}$ is not good to manage inputs that have to be routinely repeated. We fix these problems straight away by defining a new category as follows:

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

- Obj $\mathbb{B}^k_{\boldsymbol{ZKP}} :=$ Obj $\mathbb{B}_{\boldsymbol{circ}}$*;*
- Mor $\mathbb{B}^k_{\boldsymbol{ZKP}}(A,B) :=$ Mor $\mathbb{B}_{\boldsymbol{circ}}(A \otimes X^k \otimes X^n, X \otimes B)$*, 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:*



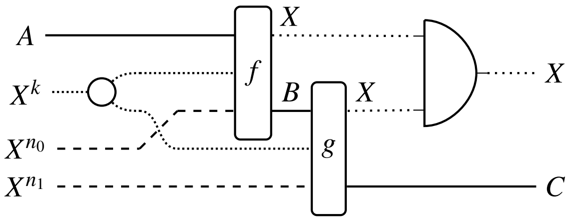
- $id_A := \top \otimes id_A \otimes id_{X^k} : A \otimes X^k \otimes X^0 \to X \otimes A$*. Identites are depicted as follows:*



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

$$(id_A \otimes {-\!\circ\!\!<}_{X^k} \otimes id_{X^{n_0} \otimes X^{n_1}});(id_{A \otimes X^k} \otimes \sigma_{X^{n_0},X^k} \otimes id_{X^{n_1}});(f \otimes id_{X^k \otimes X^{n_1}});(id_X \otimes g);(\rhd\!\!- \otimes id_C)$$

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



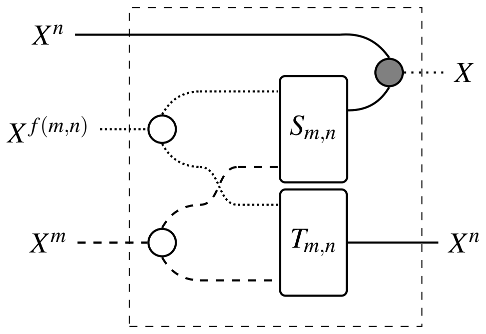
In words, we compose morphisms by wiring the dotted wires together into an `AND` *gate, by considering the monoidal product of the dashed wires as the dashed wire of the composition, and by feeding a copy the densely dotted input to both circuits.*

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

Proceeding as in Lemma 4.2, we are able to prove the following theorem:

Theorem 5.2. *For each n,m, denote with $f(m,n)$ 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* ***Count*** $\to \mathbb{B}^{f(m,n)}_{\mathbf{ZKP}}$ *sending each number k to the k-fold composition of the morphism*

$$(id_{X^n} \otimes -\!\circ\!\!\prec_{X^{f(m,n)}} \otimes -\!\circ\!\!\prec_{X^m}); (id_{X^n \otimes X^{f(m,n)}} \otimes \sigma_{X^{f(m,n)},X^m} \otimes id_{X^m}); (id_{X^n} \otimes S_{m,n} \otimes T_{m,n}); (\succ\!\!\bullet\cdots_{X^n} \otimes id_{X^n})$$



Notice that Theorem 5.2 is stronger than one would expect: As in Theorem 4.3 we obtained a circuit which not only operated on paths of length n, but also on all paths of smaller length for a given graph G, here we have something similar: $f(m,n)$ allows us to feed to the circuit the specification of *any* graph that has *up to m* edges and n vertexes! In this sense, fixing m,n amounts to fix some upper bounds for the size of the graph, exactly as taking the k-fold composition of the circuit above amounts to fix some upper bound on the size of the path we want to process.

Proceeding as in Section 4.3, we can define a snarkizator for the category $\mathbb{B}^{f(m,n)}_{\mathbf{ZKP}}$ by trivially adapting Definition 4.4. *A snarkized k-fold composition of the circuit above verifies if a sequence of k or less edges in any graph having at most m edges and n vertexes constitutes a valid path in the graph or not.* This is exactly what we wanted: A zk-SNARK built on such circuit can be used to succintly proved that a given piece of data constitutes a valid path in the state graph of a specified finite state machine. In other words, *such zk-SNARK verifies that the rules specified by a given FSM have been followed.*

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

Theorem 5.3. *Let G, G' be graphs with n,n' vertices and m,m' edges, respectively. Denote with $f(m,n)$ the function outputting how many bits are needed to store the source and target truth tables for graphs with n vertexes and m edges. Then for each morphism $G \to G'$ the following diagram commutes:*

$$\begin{tikzcd}
 & & & \mathbb{B}^{f(m,n)}_{\textbf{\textit{ZKP}}} \arrow[rrr, equal] \arrow[dd] & & & \mathbb{B}^{f(m,n)}_{\textbf{\textit{ZKP}}} \arrow[dd] \\
 & & \textbf{\textit{Count}} \arrow[ru] \arrow[rrr] \arrow[ddd, equal] & & & \mathbb{B}^{G}_{\textbf{\textit{Graph}}} \arrow[ru] \arrow[ddd] & \\
 & & & \mathbb{B}^{f(m',n')}_{\textbf{\textit{ZKP}}} \arrow[rrr, equal] & & & \mathbb{B}^{f(m',n')}_{\textbf{\textit{ZKP}}} \\
G \arrow[r] \arrow[dd] & \mathfrak{F}(G) \arrow[ruu] \arrow[rrr] \arrow[dd] & & & \mathbb{B}^{G}_{\textbf{\textit{path}}} \arrow[ruu] \arrow[dd] & & \\
 & & \textbf{\textit{Count}} \arrow[ruu] \arrow[rrr] & & & \mathbb{B}^{G'}_{\textbf{\textit{Graph}}} \arrow[ruu] & \\
G' \arrow[r] & \mathfrak{F}(G') \arrow[ru] \arrow[rrr] & & & \mathbb{B}^{G'}_{\textbf{\textit{path}}} \arrow[ru] & &
\end{tikzcd}$$

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, with each zk-SNARK verifying that the rules specified by a FSM have been followed.

Ongoing work includes implementing our correspondence in a formally verified setting using dependent types. To do this, we are using `idris-ct` [16], our own library to do category theory in a dependently typed framework (`Idris` [3]).

Future work is mainly focused in generalizing our machinery to map free symmetric strict monoidal categories into boolean circuits, providing a way to define circuits verifying executions for Petri nets. Major challenges for this task revolve around the fact that the number of tokens in a Petri net marking can be *unbounded*. This proves necessary to rethink the way we store object-related information in a boolean circuit.

Another interesting line of research revolves around using *recursive zk-SNARKs* [2] to extend the verifying capacities of zk-SNARKs beyond a previously fixed upper bound for the edges that can compose a graph path. The main idea is that to verify, say, that $2n$ edges form a valid path in a graph G, we can use a zk-SNARK verifying that the first n edges form a path in G, and recursively feed it to a zk-SNARK verifying that the last n edges form a path in G. This recursive SNARK then verifies that the overall sequence of $2n$ edges is a valid path in G. The possibility of recursively composing zk-SNARKs seems very promising to generalize our strategy to the verification of paths of arbitrary length.

Acknowledgements

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

References

[1] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer & Madars Virza: *SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge*. In Ran Canetti & Juan A. Garay, editors: *Advances in Cryptology – CRYPTO 2013*, 8043, Springer Berlin Heidelberg, pp. 90–108, doi:10.1007/978-3-642-40084-1_6. Available at `http://link.springer.com/10.1007/978-3-642-40084-1_6`. (Cited on page 1.)

[2] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer & Madars Virza: *Scalable Zero Knowledge Via Cycles of Elliptic Curves* 79(4), pp. 1102–1160. doi:10.1007/s00453-016-0221-0. (Cited on pages 1 and 10.)

[3] Edwin Brady: *Idris, a General-Purpose Dependently Typed Programming Language: Design and Implementation* 23(05), pp. 552–593. doi:10.1017/S095679681300018X. (Cited on page 10.)

[4] Vitalik Buterin: *A Next-Generation Smart Contract and Decentralized Application Platform*, pp. 1–36. Available at `http://buyxpr.com/build/pdfs/EthereumWhitePaper.pdf`. (Cited on page 1.)

[5] Rosario Gennaro, Craig Gentry, Bryan Parno & Mariana Raykova (2013): *Quadratic Span Programs and Succinct NIZKs without PCPs*. 7881, doi:10.1007/978-3-642-38348-9_37. (Cited on page 8.)

[6] Fabrizio Genovese, Alex Gryzlov, Jelle Herold, Marco Perone, Erik Post & André Videla: *Computational Petri Nets: Adjunctions Considered Harmful*. Available at `http://arxiv.org/abs/1904.12974`. (Cited on pages 2 and 4.)

[7] Herbert B. Henderton: *A Mathematical Introduction to Logic*, 2nd edition. Academic Press, doi:10.1016/C2009-0-22107-6. Available at `https://linkinghub.elsevier.com/retrieve/pii/C20090221076`. (Cited on page 2.)

[8] Daira Hopwood, Sean Bowe, Taylor Hornby & Nathan Wilcox: *Zcash Protocol Specification*. Available at `https://github.com/puffnfresh/iridium`. (Cited on page 1.)

[9] Oded Leiba, Yechiav Yitzchak, Ron Bitton, Asaf Nadler & Asaf Shabtai: *Incentivized Delivery Network of IoT Software Updates Based on Trustless Proof-of-Distribution*. Available at `http://arxiv.org/abs/1805.04282`. (Cited on page 1.)

[10] Saunders MacLane: *Categories for the Working Mathematician*. Graduate Texts in Mathematics 5, Springer New York, doi:10.1007/978-1-4757-4721-8. Available at `http://link.springer.com/10.1007/978-1-4757-4721-8`. (Cited on page 4.)

[11] Richard MacManus: *Does blockchain size matter?* Available at `https://blocksplain.com/2018/02/22/blockchain-size/`. (Cited on page 1.)

[12] Anastasia Mavridou & Aron Laszka: *Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach*. Available at `http://arxiv.org/abs/1711.09327`. (Cited on page 1.)

[13] Satoshi Nakamoto: *Bitcoin: A Peer-to-Peer Electronic Cash System*, pp. 1–9. Available at `https://bitcoin.org/bitcoin.pdf`. (Cited on page 1.)

[14] Grigore Rosu: *Formal Design, Implementation and Verification of Blockchain Languages (Invited Talk)*. doi:10.4230/lipics.fscd.2018.2. (Cited on page 1.)

[15] Scipr-lab: *Libsnark Github Page*. Available at `https://github.com/scipr-lab/libsnark`. (Cited on page 7.)

[16] Statebox Team: *Idris-Ct Github Page*. Available at `https://github.com/statebox/idris-ct`. (Cited on page 10.)

[17] Statebox Team: *The Mathematical Specification of the Statebox Language*. Available at `http://arxiv.org/abs/1906.07629`. (Cited on page 4.)

[18] Heribert Vollmer: *Introduction to Circuit Complexity*. Texts in Theoretical Computer Science An EATCS Series, Springer Berlin Heidelberg, doi:10.1007/978-3-662-03927-4. Available at `http://link.springer.com/10.1007/978-3-662-03927-4`. (Cited on page 18.)

Appendix - Proofs

Theorem 2.4. *There exists a strict monoidal functor* $\mathtt{ext}^G : \mathbb{B}^G_{\mathit{circ}} \to \mathbb{B}_{\mathit{fun}}$ *sending the generating morphisms* m_g *to the function* $\mathtt{int}(g)$.

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

Lemma .1. $\mathbb{B}_{\mathit{KP}}$ *as defined in Definition 2.7 is a bicategory.*

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

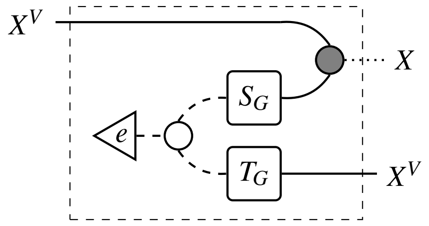
Moreover, composition is clearly functorial. This follows from the fact that extensionality is a congruence with respect to function composition.

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

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

Theorem 4.1. *Having chosen an enumeration on the vertexes and edges of a graph G, there is a pseudofunctor* $\mathfrak{F}(G) \to \mathbb{B}_{\mathit{KP}}$, *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:*

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



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

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

Lemma 4.1. *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 $\mathfrak{F}(G)$. Moreover, observe how any morphism of graphs induces a functor between their corresponding free categories which sends generating morphisms to generating morphisms. In particular, this means that such functor preserves the length of paths, making the following diagram trivially commute:

$$\begin{tikzcd}
G \arrow[r] \arrow[d, "f"'] & \mathfrak{F}(G) \arrow[r] \arrow[d, "\mathfrak{F}(f)"'] & \textbf{Count} \arrow[d, equal] \\
G' \arrow[r] & \mathfrak{F}(G') \arrow[r] & \textbf{Count}
\end{tikzcd}$$

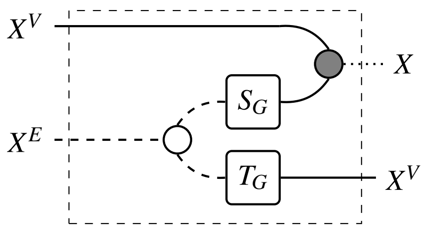
This suffices to prove the existence of a functorial correspondence $\textbf{Graph} \to \textbf{Cat}(\textbf{Count}, \textbf{Count})$. $\square$

Theorem 4.3. *For a graph G, consider an enumeration and S_G and T_G as defined in Theorem 4.1. There is a pseudofunctor $\textbf{Count} \to \mathbb{B}_{\textbf{KP}}$ sending $*$ to X^V, 0 to id_{X^V} and $n > 0$ to the n-fold composition of the morphism*

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

The composition of this pseudofunctor with the pseudofunctor of Lemma 4.2 gives a pseudofunctor $\mathfrak{F}(G) \to \textbf{Count} \to \mathbb{B}_{\textbf{KP}}$ sending each object to X^V, and each generating morphism to the circuit:

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



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

Proof. The only non-trivial part is proving pseudofunctoriality of the functor $\textbf{Count} \to \mathbb{B}_{\textbf{KP}}$. 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 $\mathbb{B}_{\textbf{KP}}$, 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 $\mathbb{B}_{\textbf{KP}}$ is itself trivial. $\square$

Lemma .2. *For each n, $\mathbb{B}^n_{\textbf{ZKP}}$ 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 $-\!\!\circ\!\!<^m_X$ and $\sigma_{X^{f(n,m)},X^m}$. $\square$

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

$$\begin{tikzcd}
 & & & \mathbb{B}^{f(m,n)}_{\textbf{\textit{ZKP}}} \arrow[rrr, equal] \arrow[dd] & & & \mathbb{B}^{f(m,n)}_{\textbf{\textit{ZKP}}} \arrow[dd] \\
 & & \textbf{\textit{Count}} \arrow[rrr] \arrow[ur] \arrow[ddd, equal] & & & \mathbb{B}^{G}_{\textbf{\textit{Graph}}} \arrow[ur] \arrow[ddd] & \\
 & & & \mathbb{B}^{f(m',n')}_{\textbf{\textit{ZKP}}} \arrow[rrr, equal] & & & \mathbb{B}^{f(m',n')}_{\textbf{\textit{ZKP}}} \\
G \arrow[r] \arrow[dd] & \mathfrak{F}(G) \arrow[rrr] \arrow[ur] \arrow[dd] & & & \mathbb{B}^{G}_{\textbf{\textit{path}}} \arrow[ur] \arrow[dd] & & \\
 & & \textbf{\textit{Count}} \arrow[rrr] \arrow[uur] & & & \mathbb{B}^{G'}_{\textbf{\textit{Graph}}} \arrow[uur] & \\
G' \arrow[r] & \mathfrak{F}(G') \arrow[rrr] \arrow[ur] & & & \mathbb{B}^{G'}_{\textbf{\textit{path}}} \arrow[ur] & &
\end{tikzcd}$$

Proof. We will prove the commutativity of single squares, starting from the top face. In the square:

$$\begin{tikzcd}
\textbf{Count} \arrow[r] & \mathbb{B}^{G}_{\textbf{Graph}} \\
\mathfrak{F}(G) \arrow[r] \arrow[u] & \mathbb{B}^{G}_{\textbf{path}} \arrow[u]
\end{tikzcd}$$

We notice that the bottom functor acts by sending each edge e to the morphism:

$$(id_{X^n} \otimes e);(id_{X^n} \otimes -\!\circ\!\!<_{X^m});(id_{X^n} \otimes S_G \otimes T_G);(>\!\!\bullet\cdots_{X^n} \otimes id_{X^n})$$

Being a subcategory of $\mathbb{B}_{\textbf{circ}}$, which is free symmetric monoidal, eavery morphism in $\mathbb{B}^{G}_{\textbf{path}}$ is just a composition of a finite number of morphisms as the one above for some edges $e_1, \dots, e_n$. We then define a pseudofunctor sending each morphism

$$(id_{X^n} \otimes e);(id_{X^n} \otimes -\!\circ\!\!<_{X^m});(id_{X^n} \otimes S_G \otimes T_G);(>\!\!\bullet\cdots_{X^n} \otimes id_{X^n})$$

To:

$$(id_{X^n} \otimes -\!\circ\!\!<_{X^m});(id_{X^n} \otimes S_G \otimes T_G);(>\!\!\bullet\cdots_{X^n} \otimes id_{X^n})$$

The mapping on 2-cells is trivial. Pseudofunctoriality is obvious and follows from function extensionality being a congruence wrt function composition. Commutativity is obvious too since the morphism above is precisely where precisely where each generating morphism in $\mathfrak{F}(G)$ gets sent to when going through **Count**.

The square:

$$\begin{tikzcd}
\mathbb{B}^{f(n,m)}_{\mathbf{ZKP}} \arrow[r, equal] & \mathbb{B}^{f(n,m)}_{\mathbf{ZKP}} \\
\mathbf{Count} \arrow[u] \arrow[r] & \mathbb{B}^{G}_{\mathbf{Graph}} \arrow[u]
\end{tikzcd}$$

Commutes by noticing that $\mathbb{B}^{G}_{\mathbf{Graph}}$ is again generated by the morphism

$$(id_{X^n} \otimes -\!\!\circ\!\!<_{X^m}); (id_{X^n} \otimes S_G \otimes T_G); (>\!\!\!\bullet\!\cdots_{X^n} \otimes id_{X^n})$$

We can then define the right-edge pseudofunctor to be sending it to:

$$(id_{X^n} \otimes -\!\!\circ\!\!<_{X^{f(n,m)}} \otimes -\!\!\circ\!\!<_{X^m}); (id_{X^n \otimes X^{f(n,m)}} \otimes \sigma_{X^{f(n,m)},X^m} \otimes id_{X^m}); (id_{X^n} \otimes S_{n,m} \otimes T_{n,m}); (>\!\!\!\bullet\!\cdots_{X^n} \otimes id_{X^n})$$

The mapping on 2-cells is again trivial. Pseudofunctoriality and square commutativity follow as in the previous case.

The bottom face is equal to the top one, so we now switch to the side faces. Consider a graph morphism $g : G \to G'$. Commutativity of

$$\begin{tikzcd}
G \arrow[r] \arrow[d, "g"'] & \mathfrak{F}(G) \arrow[r] \arrow[d, "\mathfrak{F}(g)"'] & \mathbf{Count} \arrow[d, equal] \\
G' \arrow[r] & \mathfrak{F}(G') \arrow[r] & \mathbf{Count}
\end{tikzcd}$$

Is just Lemma 4.2. As for the square:

$$\begin{tikzcd}
\mathbf{Count} \arrow[r] \arrow[d, equal] & \mathbb{B}^{f(n,m)}_{\mathbf{ZKP}} \arrow[d] \\
\mathbf{Count} \arrow[r] & \mathbb{B}^{f(n',m')}_{\mathbf{ZKP}}
\end{tikzcd}$$

It is sufficient to define the pseudofunctor $\mathbb{B}^{f(n,m)}_{\mathbf{ZKP}} \to \mathbb{B}^{f(n',m')}_{\mathbf{ZKP}}$ as sending X^n to $X^{n'}$, and the generating morphism

$$(id_{X^n} \otimes -\!\!\circ\!\!<_{X^{f(n,m)}} \otimes -\!\!\circ\!\!<_{X^m}); (id_{X^n \otimes X^{f(n,m)}} \otimes \sigma_{X^{f(n,m)},X^m} \otimes id_{X^m}); (id_{X^n} \otimes S_{n,m} \otimes T_{n,m}); (>\!\!\!\bullet\!\cdots_{X^n} \otimes id_{X^n})$$

To:

$$(id_{X^{n'}} \otimes -\!\!\circ\!\!<_{X^{f(n',m')}} \otimes -\!\!\circ\!\!<_{X^{m'}}); (id_{X^{n'} \otimes X^{f(n',m')}} \otimes \sigma_{X^{f(n',m')},X^{m'}} \otimes id_{X^{m'}}); (id_{X^{n'}} \otimes S_{n',m'} \otimes T_{n',m'}); (>\!\!\!\bullet\!\cdots_{X^{n'}} \otimes id_{X^{n'}})$$

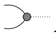
According to this definition extensionally equal circuits are sent to extensionally equal circuits, so 2-cells can be defined in the obvious way. Commutativity of the square is again true by definition. A slight modification of the proof above allows us to also prove the commutativity of the following square:

$$\begin{tikzcd}
\textbf{Count} \arrow[r] \arrow[d, equal] & \mathbb{B}^{G}_{\textbf{Graph}} \arrow[d] \\
\textbf{Count} \arrow[r] & \mathbb{B}^{G'}_{\textbf{Graph}}
\end{tikzcd}$$

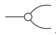
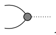
Now we focus on:

$$\begin{tikzcd}
G \arrow[r] \arrow[d, "g"'] & \mathfrak{F}(G) \arrow[r] \arrow[d, "\mathfrak{F}(g)"'] & \mathbb{B}^{G}_{\textbf{path}} \arrow[d] \\
G' \arrow[r] & \mathfrak{F}(G') \arrow[r] & \mathbb{B}^{G'}_{\textbf{path}}
\end{tikzcd}$$

Whose commutativity is obvious by defining the pseudofunctor $\mathbb{B}^{G}_{\textbf{path}} \to \mathbb{B}^{G'}_{\textbf{path}}$ by sending X^n to $X^{n'}$, and every morpsism

$(id_{X^n} \otimes e);(id_{X^n} \otimes$ $_{X^m});(id_{X^n} \otimes S_G \otimes T_G);($  $_{X^n} \otimes id_{X^n})$

To:

$(id_{X^{n'}} \otimes \mathfrak{F}(g)(e));(id_{X^{n'}} \otimes$  $_{X^{m'}});(id_{X^{n'}} \otimes S_G \otimes T_G);($  $_{X^{n'}} \otimes id_{X^{n'}})$

2-cells mapping is again obvious.

Finally, we focus on the squares:

$$\begin{tikzcd}
\mathbb{B}^{G}_{\textbf{path}} \arrow[r] \arrow[d] & \mathbb{B}^{G}_{\textbf{Graph}} \arrow[d] & \mathbb{B}^{G}_{\textbf{Graph}} \arrow[r] \arrow[d] & \mathbb{B}^{f(m,n)}_{\textbf{ZKP}} \arrow[d] \\
\mathbb{B}^{G'}_{\textbf{path}} \arrow[r] & \mathbb{B}^{G'}_{\textbf{Graph}} & \mathbb{B}^{G'}_{\textbf{Graph}} \arrow[r] & \mathbb{B}^{f(m',n')}_{\textbf{ZKP}}
\end{tikzcd}$$

Whose commutativity is proven similarly. All pseudofunctors involved in these squares have already been defined previously. Commutativity is obvious by tracking where generating morphisms get mapped. □

Appendix - Building the morphisms $S_G, T_G, S_{m,n}, T_{m,n}$

In Section 4 we mentioned a couple of boolean functions, $S_G, T_G : \mathbb{B}^E \to \mathbb{B}^V$: They depend on having fixed a graph G, and return as output the enumeration of a source and target vertex, respectively, of an edge whose enumeration is specified as input.

The beauty of category theory lies precisely in the fact that we were able to abstract on how S_G and T_G are made, and just focusing on how these functions had to be used to assemble the whole verifying circuit for G. Nevertheless, a precise description of these functions is still needed to make an implementation possible, which we are going to provide in this appendix. We warn the reader that our solution is far from being optimal in terms of complexity, so it should be understood as a proof of concept.

In the following, we will explain the process for the function S_G, the one for T_G being similar. First, we build S_G as a table using the incidency matrix of the graph G.

Edge	Source Vertex
id_{v_1}	v_1
$\dots$	$\dots$
id_{v_n}	v_n
e_1	$s(e_1)$
$\dots$	$\dots$
e_m	$s(e_m)$
u_1	$\mathbf{0}$
$\dots$	$\dots$
u_k	$\mathbf{0}$

Here $s(\_)$ represents the source function associated to G. From here, we switch to enumerations, where we denote with x_j^i the i-th binary digit of the enumeration of element x_j:

Edge 1st bit		Edge E-th bit	Source 1st bit		Source V-th bit
$id_{v_1}^1$	$\dots$	$id_{v_1}^E$	v_1^1	$\dots$	v_1^V
	$\dots$			$\dots$	
$id_{v_n}^1$	$\dots$	$id_{v_n}^E$	v_n^1	$\dots$	v_n^V
e_1^1	$\dots$	e_1^E	$s(e_1)^1$	$\dots$	$s(e_1)^V$
	$\dots$			$\dots$	
e_m^1	$\dots$	e_m^E	$s(e_m)^1$	$\dots$	$s(e_m)^V$
u_1^1	$\dots$	u_1^E	0	$\dots$	0
	$\dots$			$\dots$	
u_k^1	$\dots$	u_k^E	0	$\dots$	0

We now split the table into V different tables, each returning only one digit of the vertex enumeration:

Edge 1st bit		Edge E-th bit	Source 1st bit
$id_{v_1}^1$	$\dots$	$id_{v_1}^E$	v_1^1
	$\dots$		
$id_{v_n}^1$	$\dots$	$id_{v_n}^E$	v_n^1
e_1^1	$\dots$	e_1^E	$s(e_1)^1$
	$\dots$		
e_m^1	$\dots$	e_m^E	$s(e_m)^1$
u_1^1	$\dots$	u_1^E	0
	$\dots$		
u_k^1	$\dots$	u_k^E	0

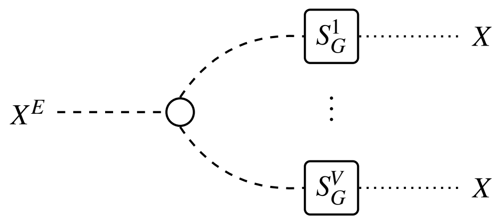
$\dots$

Edge 1st bit		Edge E-th bit	Source V-th bit
$id_{v_1}^1$	$\dots$	$id_{v_1}^E$	v_1^V
	$\dots$		
$id_{v_n}^1$	$\dots$	$id_{v_n}^E$	v_n^V
e_1^1	$\dots$	e_1^E	$s(e_1)^V$
	$\dots$		
e_m^1	$\dots$	e_m^E	$s(e_m)^V$
u_1^1	$\dots$	u_1^E	0
	$\dots$		
u_k^1	$\dots$	u_k^E	0

Using standard techniques [18] we are able to turn each one of these tables into a digital circuit. We then obtain a bunch of morphisms in $\mathbb{B}_{\mathbf{circ}}$:

$$\begin{tikzcd}
X^E \arrow[r, dash, dashed] & \boxed{S_G^1} \arrow[r, dash, dotted] & X & \cdots & X^E \arrow[r, dash, dashed] & \boxed{S_G^V} \arrow[r, dash, dotted] & X
\end{tikzcd}$$

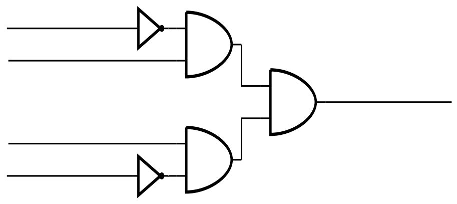
The resulting morphism S_G is obtained by just copying the inputs and tensoring the morphisms together:



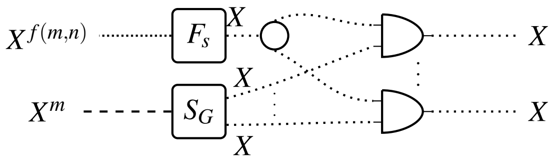
Here one should note that the V-fold monoidal product of X is just X^V, so the string diagram above is a morphism of the right type, from X^E to X^V.

Now we focus on the morphisms $S_{m,n}, T_{m,n} : X^{f(m,n)} \otimes X^m \to X^n$. These morphisms accept a graph enumeration of $f(m,n)$ bits in input, along with an edge enumeration of m bits, and return a vertex enumeration of n bits, representing the source and target, respectively, of the edge provided in the specified graph.

Again, we focus on $S_{m,n}$, the procedure for $T_{m,n}$ being similar. We start by noticing that for a fixed bitstring s of length $f(m,n)$, we can produce a circuit $F_s : X^{f(m,n)} \to X$ (called *filter for s*) that outputs 1 only if the input is s, and 0 otherwise: For each bit of s, we concatenate it with a NOT gate if it is 0, and we leave it as it is otherwise. Then we wire all the bits with AND ports. For instance, the circuit below, working for inputs of 4 bits, outputs 1 only if the input is 1001.

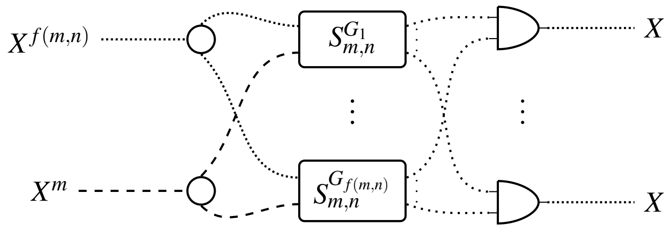


Interpreting a bitstring s as the enumeration of a graph G with m edges and n vertexes, we can consider the circuit:



That is, we are connecting the output of F_s with each output bit of S_G using AND ports. The result is a gate $S_{m,n}^G : X^{f(m,n)} \otimes X^m \to X^n$ that acts like S_G only when the input on $X^{f(n,m)}$ is the enumeration of the graph G, and outputs 0 in any other case.

The circuit $S_{m,n}$ is obtained by tensoring together all the $S_{m,n}^G$ gates for each graph G of m edges and v vertexes, by copying the inputs and taking the OR of the outputs.



$X^{f(m,n)}$
$S_{m,n}^{G_1}$
X
X^m
$S_{m,n}^{G_{f(m,n)}}$
X