Geometry of Interaction for ZX-Diagrams

Kostia Chardonnet, Benoît Valiron, Renaud Vilmart

▶ **To cite this version:**

Kostia Chardonnet, Benoît Valiron, Renaud Vilmart. Geometry of Interaction for ZX-Diagrams. MFCS 2021 - 46th International Symposium on Mathematical Foundations of Computer Science, Aug 2021, Tallinn, Estonia. pp.30:1–30:16, 10.4230/LIPIcs.MFCS.2021.30 . hal-03154573

HAL Id: hal-03154573

https://hal.science/hal-03154573

Submitted on 1 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Geometry of Interaction for ZX-Diagrams

Kostia Chardonnet
LMF – Université Paris Saclay / IRIF – Université de Paris, France

Benoît Valiron
LMF, CentraleSupélec, Université Paris Saclay, France

Renaud Vilmart
LMF, Inria, Université Paris-Saclay, France

Abstract

ZX-Calculus is a versatile graphical language for quantum computation equipped with an equational theory. Getting inspiration from Geometry of Interaction, in this paper we propose a token-machine-based asynchronous model of both pure ZX-Calculus and its extension to mixed processes. We also show how to connect this new semantics to the usual standard interpretation of ZX-diagrams. This model allows us to have a new look at what ZX-diagrams compute, and give a more local, operational view of the semantics of ZX-diagrams.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Linear logic; Theory of computation → Equational logic and rewriting

Keywords and phrases Quantum Computation, Linear Logic, ZX-Calculus, Geometry of Interaction

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Quantum computing is a model of computation where data is stored on the state of particles governed by the law of quantum physics. The theory is well established enough to have allowed the design of quantum algorithms whose applications are gathering interests from both public and private actors [34, 36, 16] Together with the progresses in physical capabilities, quantum computers are envisioned as a disruptive technology in the coming years [28].

One of the fundamental properties of quantum objects is to have a *dual* interpretations. In the first one, the quantum object is understood as a *particle*: with a definite, localized point in space, distinct from the other particles. Light can be for instance regarded as a set of photons. In the other interpretation, the object is understood as a *wave*: it is "spread-out" in space, possibly featuring interference. This is for instance the interpretation of light as an electromagnetic wave.

The standard model of computation uses *quantum bits* (qubits) for storing information and *quantum circuits* [35] for describing quantum operations with *quantum gates*, the quantum version of Boolean gates. In this model, on one hand quantum bits are intuitively seen as tokens flowing inside the wires of the circuit. On the other hand, the *state* of all of the quantum bits of the memory is mathematically represented as a vector in a (finite dimensional) Hilbert: the set of quantum bits is a wave flowing in the circuit, from the inputs to the output, while the computation generated by the list of quantum gates is a *linear map* from the Hilbert space of inputs to the Hilbert space of outputs. Although the pervasive model for quantum computation, quantum circuits' operational semantics is only given in an intuitive manner. A quantum circuit informally describes a series of "gate applications", akin to some sequential, low-level assembly language where quantum gates are opaque black-boxes.

Quantum circuits do not feature any native formal operational semantics giving rise to abstract reasoning, equational theory or well-founded rewrite system. To be able to reason on quantum circuits, until recently the only choice was to rely on the unitary-matrix semantics

© Kostia Chardonnet and Benoît Valiron and Renaud Vilmart;
licensed under Creative Commons License CC-BY 4.0
42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:25
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of circuits. However, because the dimension of the matrix corresponding to a circuit is exponential on the number of qubits involved, this solution is very expensive and limited to simple cases.

To bring some scalability to the approach, a recent proposal is sum-over-path semantics [1, 6]. Still based on the original mathematical representation of state-as-a-vector, the sum-over-path of a quantum circuit synthesizes the operation described by the circuit into a few simple constructs: a Boolean operation as action on the basis states, and a so-called *phase polynomial*, bringing to circuits a formal flavor of *wave-style semantics*.

The main line of work formalizing a token-based operational semantics for quantum circuit [32] is based on Geometry of Interaction (GoI) [20, 19, 18, 21, 23, 24, 25]. Among its many instantiations, GoI can be seen as a procedure to intepret a proof-nets [22] —graphical representation of proofs of linear logic [17]— as a token-based automaton [9, 2]. The flow of a token inside a proof-net characterizes an invariant of the proof —its computational content. This framework is used in [32] to formalize the notion of qubits-as-tokens flowing inside a higher-order term representing a quantum computation —that is, computing a quantum circuit. However, in this work, quantum gates are still regarded as black-boxes, and tokens are purely classical objects requiring synchronicity: to fire, a two-qubit gate needs its two arguments to be ready.

In recent years, an alternative model of quantum computation with better formal properties has however emerged: the ZX calculus [7]. Originally motivated by a categorical interpretation of quantum theory, the ZX-Calculus is a graphical language that represents linear maps as special kinds of graphs called *diagrams*. The calculus comes with a well-defined equational theory making it possible to reason on quantum computation by means of local graph rewriting. Unlike the qunatum circuit framework, ZX-Calculus also comes with, a small set of canonical generators with a well-defined semantics.

Reasonning about ZX can therefore be done in two ways: with the linear operator semantics (aka matrix semantics), or through graph rewriting. This graphical language has been shown to be amenable to many extensions and is being used in a wide spectrum of applications ranging from quantum circuit optimization [13, 4], verification [27, 14, 12] and representation such as MBQC patterns [15] or error-correction [11, 10].

As a summary, despite their ad-hoc construction, quantum circuits can be seen from two perspectives: computation as a flow of particles (i.e. tokens), and as a wave passing through the gates. On the other hand, although ZX-Calculus is a well-founded language, it still misses such a perspective.

In this paper, we aim at providing ZX with a particle-style and a wave-style semantics, similarly to what has been done for quantum circuits.

Following the idea of applying a token machine to proof-nets in order to study its computational content, we present in this paper a token machine for the ZX-Calculus and its extension to mixed processes [8, 5]. We show how it links to the standard interpretation of ZX-diagrams. While the standard interpretation of ZX-diagrams proceeds with conventional graph rewriting, the tokens flowing inside the diagram do not modify it, and the computation emerges from their ability to enter into superposition. We derive two perspectives on this phenomenon: one purely token-based and one based on a sum-over-path interpretation.

Plan of the paper. The paper is organized as follows : in Section 2 we present the ZX-Calculus and its standard interpretation into **Qubit**, and its axiomatization.

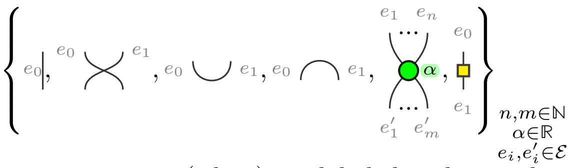
In Section 3 we present the actual asynchronous token machine and its semantics and show that it is sound and complete with regard to the standard interpretation of ZX-diagrams. We then modify it in Section 4 to use a Sum-Over-Path interpretation in order to avoid an exponential blow up in the number of state in our Token Machine. Next, in Section 5 we present an extension of the ZX-Calculus to mixed processes and adapt the token machine to take this extension into account. Finally, in Section 6 we discuss synchronicity and other ways to represent the Token Machine. Proofs are in the appendix.

2 The ZX-Calculus

The ZX-Calculus is a powerful graphical language for reasoning about quantum computation introduced by Bob Coecke and Ross Duncan [7]. A term in this language is a graph —called a *string diagram*— built from a core set of primitives. In the standard interpretation of ZX-Calculus, a string diagram is interpreted as a matrix. The language is equipped with an equational theory preserving the standard interpretation.

2.1 Pure Operators

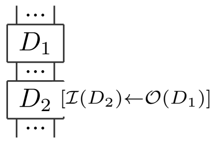
The so-called *pure* ZX-diagrams are generated from a set of primitives, given on the right: the Identity, Swap, Cup, Cap, Green-spider and H-gate:



We shall be using the following labeling convention: wires (edges) are labeled with e_i, taken from an infinite set of labels $\mathcal{E}$. We take for granted that distinct wires have distinct labels. The real number α attached to the green spiders is called the *angle*. ZX-diagrams are read top-to-bottom: dangling top edges are the *input edges* and dangling edges at the bottom are *output edges*. For instance, Swap has 2 input and 2 output edges, while Cup has 2 input edges and no output edges. We write $\mathcal{E}(D)$ for the set of edge labels in the diagram D, and $\mathcal{I}(D)$ (resp. $\mathcal{O}(D)$) for the list of input edges (resp. output edges) of D. We denote :: the concatenation of lists.

ZX-primitives can be composed as follows.

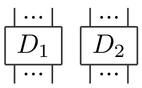
Sequentially If $\mathcal{E}(D_1)\cap\mathcal{E}(D_2)=\emptyset$, then:

- $D_2 \circ D_1 :=$ 

- $\mathcal{E}(D_2\circ D_1)=\mathcal{E}(D_1)\cup\mathcal{E}(D_2)\setminus\mathcal{I}(D_2)$
- $\mathcal{I}(D_2\circ D_1)=\mathcal{I}(D_1)$
- $\mathcal{O}(D_2\circ D_1)=\mathcal{O}(D_2)$

Where $[\mathcal{I}(D_2)\leftarrow\mathcal{O}(D_1)]$ is the substitution of the names of the labels of $\mathcal{I}(D_2)$ by those of $\mathcal{O}(D_1)$ done left to right.

In parallel If $\mathcal{E}(D_1)\cap\mathcal{E}(D_2)=\emptyset$, then:

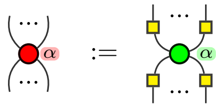
- $D_1\otimes D_2 :=$ 

- $\mathcal{E}(D_1\circ D_2)=\mathcal{E}(D_1)\cup\mathcal{E}(D_2)$
- $\mathcal{I}(D_1\otimes D_2)=\mathcal{I}(D_1) :: \mathcal{I}(D_2)$
- $\mathcal{O}(D_1\otimes D_2)=\mathcal{O}(D_1) :: \mathcal{O}(D_2)$

We write **ZX** for the set of all ZX-diagrams.

Notice that when composing diagrams with $(\_\circ\_)$, we "join" the outputs of the top diagram with the inputs of the bottom diagram. This requires that the two sets of edges have the same cardinality. The junction is then made by relabeling the input edges of the bottom diagram by the output labels of the top diagram (hence the "$[\mathcal{I}(D_2)\leftarrow\mathcal{O}(D_1)]$" in the composition).

▶ **Convention 1.** We define a second spider, red this time, by composition of Green-spiders and H-gates, as shown on the right.

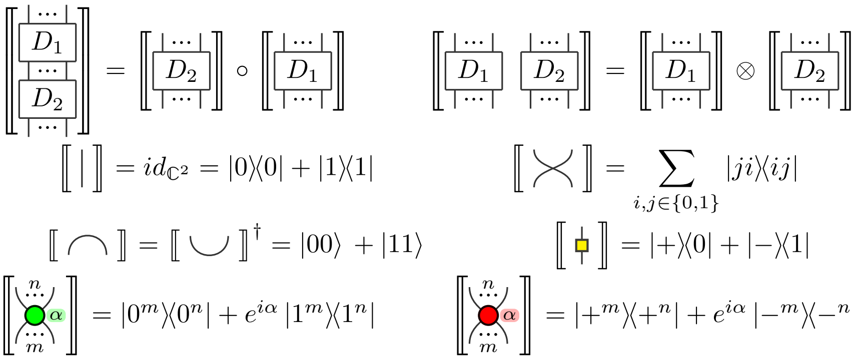


▶ **Convention 2.** We write σ for a permutation of wires, i.e any diagram generated by $\left\{ \,|\, , \times \right\}$ with sequential and parallel composition. We write the Cap as η and the Cup as ϵ. We write $Z_k^n(\alpha)$ (resp, X_k^n) for the green-node (resp, red-node) of n inputs, k outputs and parameter α and H for the H-gate. In the remainder of the paper we omit the edge labels when not necessary . Finally, by abuse of notation a green or red node with no explicit parameter holds the angle 0:

2.2 Standard Interpretation

In the *standard interpretation* [7], a diagram D is mapped to a finite dimensional Hilbert space of dimensions some powers of 2: $[\![D]\!] \in \mathbf{Qubit} := \{\mathbb{C}^{2^n} \to \mathbb{C}^{2^m} \mid n, m \in \mathbb{N}\}$.

If D has n inputs and m outputs, its interpretation is a map $[\![D]\!] : \mathbb{C}^{2^n} \to \mathbb{C}^{2^m}$ (by abuse of notation we shall use the notation $[\![D]\!] : n \to m$). It is defined inductively as follows.



Wires are interpreted with the two-dimensional Hilbert space, with orthonormal basis written as $\{|0\rangle, |1\rangle\}$, in Dirac notation [35]. Vectors of the form $|.\rangle$ (called "*kets*") are considered as vector columns, and therefore $|0\rangle = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$, $|1\rangle = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$, and $\alpha|0\rangle + \beta|1\rangle = \left(\begin{smallmatrix}\alpha\\\beta\end{smallmatrix}\right)$. Horizontal juxtaposition of wires is interpreted with the *Kronecker*, or *tensor* product. The tensor product of spaces $\mathcal{V}$ and $\mathcal{W}$ whose bases are respectively $\{v_i\}_i$ and $\{w_j\}_j$ is the vector space of basis $\{v_i \otimes w_j\}_{i,j}$, where $v_i \otimes w_j$ is a formal object consisting of a pair of v_i and w_j. We denote $|x\rangle \otimes |y\rangle$ as $|xy\rangle$. In the interpretation of spiders, we use the notation $|0^m\rangle$ to represent an m-fold tensor of $|0\rangle$. As a shortcut notation, we write $|\phi\rangle$ for column vectors consisting of a linear combinations of kets. Shortcut notations are also used for two very useful states: $|+\rangle := \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|-\rangle := \frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Dirac also introduced the notation "*bra*" $\langle x|$, standing for a row vector. So for instance, $\alpha\langle 0| + \beta\langle 1|$ is $\left(\begin{smallmatrix}\alpha & \beta\end{smallmatrix}\right)$. If $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$, we then write $\langle\phi|$ for the vector $\overline{\alpha}\langle 0| + \overline{\beta}\langle 1|$ (with $\overline{(.)}$ the complex conjugation). The notation for tensors of bras is similar to the one for kets. For instance, $\langle x| \otimes \langle y| = \langle xy|$. Using this notation, the scalar product is transparently the product of a row and a column vector: $\langle \phi \mid \psi \rangle$, and matrices can be written as sums of elements of the form $|\phi\rangle\langle\psi|$. For instance, the identity on $\mathbb{C}^2$ is $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right) = \left(\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right) + \left(\begin{smallmatrix}0&0\\0&1\end{smallmatrix}\right) = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)\left(\begin{smallmatrix}1&0\end{smallmatrix}\right) + \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)\left(\begin{smallmatrix}0&1\end{smallmatrix}\right) = |0\rangle\langle 0| + |1\rangle\langle 1|$. For more information on how Hilbert spaces, tensors, compositions and bras and kets work, we invite the reader to consult e.g. [35].

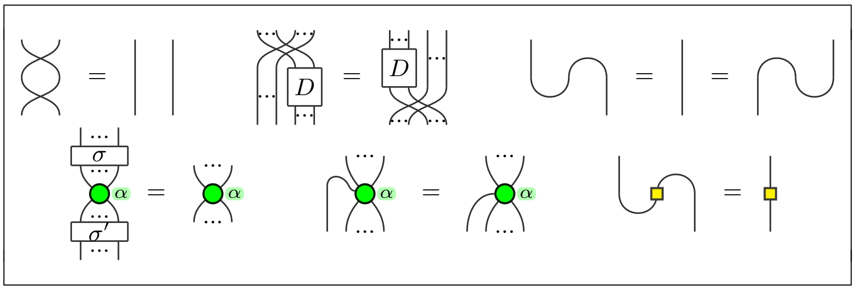


Figure 1 Connectivity rules. D represents any ZX-diagram, and σ, σ' any permutation of wires.

2.3 Properties and structure

In this section, we list several definitions and known results that we shall be using in the remainder of the paper. See e.g. [39] for more information. **Universality**. ZX-diagrams are *universal* in the sense that for any linear map $f : n \to m$, there exists a diagram D of $\mathbf{ZX}$ such that $[\![D]\!] = f$.

The price to pay for universality is that different diagrams can possibly represent the same quantum operator. There exists however a way to deal with this problem: an equational theory. Several equational theories have been designed for various fragments of the language [3, 29, 26, 30, 31, 38].

Core axiomatization. Despite this variety, any ZX axiomatization builds upon the core set of equations provided in Figure 1, meaning that edges really behave as wires that can be bent, tangled and untangled. They also enforce the irrelevance on the ordering of inputs and outputs for spiders. Most importantly, these rules preserve the standard interpretation given in Section 2.2. We will use these rules —sometimes referred to as "*only connectivity matters*"—, and the fact that they preserve the semantics extensively in the proofs of the results of the paper.

In particular, *diagrams are always considered modulo the equivalence relation presented in Figure 1.*

Completeness. The ability to transform a diagram D_1 into a diagram D_2 using the rules of some axiomatization zx (e.g. the core one presented in Figure 1) is denoted $\mathrm{zx} \vdash D_1 = D_2$.

The axiomatization is said *complete* whenever any two diagrams representing the same operator can be turned into one another using this axiomatization. Formally:

$$[\![D_1]\!] = [\![D_2]\!] \iff \mathrm{zx} \vdash D_1 = D_2$$

It is common in quantum computing to work with restrictions of quantum mechanics. Such restrictions translate to restrictions to particular sets of diagrams – e.g. the $\frac{\pi}{4}$-fragment which consists of all ZX-diagrams where the angles are multiples of $\frac{\pi}{4}$. There exist axiomatization that were proven to be complete for the corresponding fragment (all the aforementioned references tackle the problem of completeness).

The developments of this paper are given for the ZX-Calculus in its most general form, but everything in the following also works for fragments of the language.

Input and output wires. An important result which will be used in the rest of the paper is the following:

▶ **Theorem 3.** *There are isomorphisms between* $\{D \in \mathbf{ZX} \mid D : n \to m\}$ *and* $\{D \in \mathbf{ZX} \mid D : n-k \to k+m\}$ *(when* $k \leq n$*).* ◀

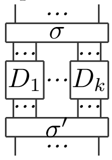
To see how this can be true, simply add cups or caps to turn input edges to output edges (or vice versa), and use the fact that we work modulo the rules of Figure 1.

When $k = n$, this isomorphism is referred to as the *map/state duality.* A related but more obvious isomorphism between ZX-diagrams is obtained by permutation of input wires (resp. output wires).

2.4 Notions of Graph Theory in ZX

Theorem 3 is essential: it allows us to transpose notions of graphs into ZX-Calculus. It is for instance possible to define a notion of connectivity.

▶ **Definition 4** (Connected Components). *Let D be a non-empty* $\mathbf{ZX}$*-diagram. Consider all of the possible decompositions with $D_1, ..., D_k \in \mathbf{ZX}$ and σ, σ' permutations of wires:*

$D =$ 

The largest such k is called the number of connected components of D. It induces a unique decomposition up to permutation of wires. The induced $D_1, ..., D_n$ are called the connected components of D. If D has only one connected component, we say that D is connected.

▶ **Definition 5** (Paths). *Let D be a ZX-diagram. A path in D between the edges e_0 and e_n is a sequence $(e_0, ..., e_n)$ of edges of D such that*

- *there exists a sequence $(g_1, ..., g_n)$ of atomic (generator) sub-diagrams of D,*
- *for $1 \leq i, j \leq n$, $g_i = g_j$ if and only if $i = j$,*
- *for $0 \leq i < n$, $e_i, e_{i+1} \in \mathcal{E}(g_{i+1})$.*

If $e_i \in \mathcal{I}(g_i)$ (resp. $e_i \in \mathcal{O}(g_{i+1})$), we say that e_i is $\uparrow$-oriented (resp. $\downarrow$-oriented) in the path.

We denote with $\mathrm{Paths}(e_0, e_n)$ *the set of paths between e_0 and e_n in D, and* $\mathrm{Paths}(D)$ *the set of all paths in D. If* $\mathrm{Paths}(e_0, e_n) = \emptyset$*, we say that e_0 and e_n are* disconnected*. Finally, the* length *of the path $p = (e_0, ..., e_n)$ is $|p| = n$.*

▶ **Definition 6** (Distance). *Let e and e' be connected edges in a ZX-diagram D. We define:*

$$d(e, e') := \min_{p \in \mathrm{Paths}(e, e')} (|p|)$$

▶ **Definition 7** (Cycles). *A cycle is defined as a path $(e_0, ..., e_n)$ where $e_0 = e_n$. We denote* $\mathrm{Cycles}(D)$ *the set of all cycles in D.*

3 A Token Machine for ZX-diagrams

Inspired by the Geometry of Interaction [20, 19, 18, 21, 23, 24, 25] and the associated notion of token machine [9, 2] for proof nets [22], we define here a first token machine on pure ZX-diagrams. The token consists of an edge of the diagram, a direction (either going up, noted $\uparrow$, or down, noted $\downarrow$) and a bit (state). The idea is that, starting from an input edge the token will traverse the graph and duplicate itself when encountering an n-ary node (such as the green and red) into each of the input / output edges of the node. Notice that it is not the case for token machines for proof-nets where the token never duplicates itself. This duplication is necessary to make sure we capture the whole linear map encoded by the ZX-diagram. Due to this duplication, two tokens might collide together when they are on the same edge and going in different directions. The result of such a collision will depend on the states held by both tokens. For a cup, cap or identity diagram, the token will simply traverse it. As for the Hadamard node the token will traverse it and become a superposition of two tokens with opposite states. Therefore, as tokens move through a diagram, some may be added, multiplied together, or annihilated.

▶ **Definition 8** (Tokens and Token States). *Let D be a ZX-diagram. A* token *in D is a triplet $(e, d, b) \in \mathcal{E}(D) \times \{\downarrow, \uparrow\} \times \{0, 1\}$. We shall omit the commas and simply write $(e\ d\ b)$. The set of tokens on D is written $\mathbf{tk}(D)$. A* token state *s is then a multivariate polynomial over $\mathbb{C}$, evaluated in $\mathbf{tk}(D)$. We define $\mathbf{tkS}(D) := \mathbb{C}[\mathbf{tk}(D)]$ the algebra of multivariate polynomials over $\mathbf{tk}(D)$.*

In the token state $t = \sum_i \alpha_i\, t_{1,i} \cdots t_{n_i,i}$, where the $t_{k,i}$'s are tokens, the components $\alpha_i\, t_{1,i} \cdots t_{n_i,i}$ are called the terms *of t.*

A monomial $(e_1\, d_1, b_1) \cdots (e_n\, d_n, b_n)$ encodes the state of n tokens in the process of flowing in the diagram D. A token state is understood as a *superposition* —a linear combination— of multi-tokens flowing in the diagram.

▶ **Convention 9.** In token states, the sum $(+)$ stands for the superposition and the product for additional tokens within a given diagram. We follow the usual convention of algebras of polynomials: for instance, if t_i stands for some token $(e_i\ d_i\ b_i)$, then $(t_1{+}t_2)t_3 = (t_1t_2){+}(t_1t_3)$, that is, the superposition of t_1,t_2 flowing in D and t_1,t_3 flowing in D. Similarly, we consider token states modulo commutativity of sum and product, so that for instance the monomial t_1t_2 is the same as t_2t_1.

3.1 Diffusion and Collision Rules

The tokens in a ZX-diagram D are meant to move inside D. The set of rules presented in this section describes an *asynchronous* evolution, meaning that given a token state, we will rewrite only one token at a time. The synchronous setting is discussed in Section 6.

▶ **Definition 10** (Asynchronous Evolution). *Token states on a diagram D are equipped with two transition systems:*

- *a* collision system *$(\rightsquigarrow_c)$, whose effect is to annihilate tokens;*
- *a* diffusion sub-system *$(\rightsquigarrow_d)$, defining the flow of tokens within D.*

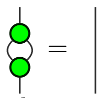
The two systems are defined as follows. With $X \in \{d, c\}$ and $1 \le j \le n_i$, if $t_{i,j}$ are tokens in $\mathbf{tk}(D)$, then using Convention 9,

$$\sum_i \alpha_i t_{i,1} \cdots t_{i,j} \cdots t_{i,n_i} \rightsquigarrow_X \sum_i \alpha_i t_{i,1} \cdots \left(\sum_k \beta_k t'_k \right) \cdots t_{i,n_i}$$

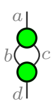
provided that $t_{i,j} \rightsquigarrow_X \sum_k \beta_k t'_k$ according to the rules of Table 1. In the table, each rule corresponds to the interaction with the primitive diagram constructor on the left-hand-side. Variables x and b span $\{0, 1\}$, and $\neg$ stands for the negation. In the green-spider rules, $e^{i\alpha x}$ stands for the the complex number $\cos(\alpha x) + i \sin(\alpha x)$ and not an edge label.

Finally, as it is customary for rewrite systems, if $(\to)$ is a step in a transition system, $(\to^*)$ stands for the reflexive, transitive closure of $(\to)$.

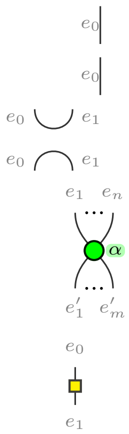
We aim at a transition system marrying both collision and diffusion steps. However, for consistency of the system, the order in which we apply them is important as illustrated by the following example.

▶ **Example 11.** Consider the equality given by the ZX equational theory:  .

If we drop a token with bit 0 at the top, we hence expect to get a single token with bit 0 at the bottom. We underline the token that is being rewriting at each step. This is what we get when giving the priority to collisions:



$$:: \qquad (a \downarrow 0) \rightsquigarrow_d \underline{(b \downarrow 0)}(c \downarrow 0) \rightsquigarrow_c (d \downarrow 0)\underline{(c \uparrow 0)(c \downarrow 0)} \rightsquigarrow (d \downarrow 0)$$



	$(e_0 \downarrow x)(e_0 \uparrow x) \rightsquigarrow_c 1$	(Positive Collision)
	$(e_0 \downarrow x)(e_0 \uparrow \neg x) \rightsquigarrow_c 0$	(Negative Collision)
	$(e_b \downarrow x) \rightsquigarrow_d (e_{\neg b} \uparrow x)$	($\cup$-diffusion)
	$(e_b \uparrow x) \rightsquigarrow_d (e_{\neg b} \downarrow x)$	($\cap$-diffusion)
	$(e_k \downarrow x) \rightsquigarrow_d e^{i\alpha x} \prod_{i \neq k} (e_i \uparrow x) \prod_{j} (e'_j \downarrow x)$ $(e'_k \uparrow x) \rightsquigarrow_d e^{i\alpha x} \prod_{j \neq k} (e'_j \downarrow x) \prod_{i} (e_i \uparrow x)$	(-Diffusion)
	$(e_0 \downarrow x) \rightsquigarrow_d (-1)^x \frac{1}{\sqrt{2}} (e_1 \downarrow x) + \frac{1}{\sqrt{2}} (e_1 \downarrow \neg x)$ $(e_1 \uparrow x) \rightsquigarrow_d (-1)^x \frac{1}{\sqrt{2}} (e_0 \uparrow x) + \frac{1}{\sqrt{2}} (e_0 \uparrow \neg x)$	(-Diffusion)

Table 1 Asynchronous token-state evolution, for all $x, b \in \{0, 1\}$

If however we decide to ignore the priority of collisions, we may end up with a non-terminating run, unable to converge to $(d \downarrow 0)$:

$$(a \downarrow 0) \rightsquigarrow_d \underline{(b \downarrow 0)}(c \downarrow 0) \rightsquigarrow_d (d \downarrow 0)\underline{(c \uparrow 0)}(c \downarrow 0) \rightsquigarrow_d (d \downarrow 0)(a \uparrow 0)(b \downarrow 0)(c \downarrow 0) \rightsquigarrow_d \dots$$

We therefore set a rewriting strategy as follows.

▶ **Definition 12** (Collision-Free). *A token state s of $\mathbf{tkS}(D)$ is called* collision-free *if:*

$$\forall s' \in \mathbf{tkS}(D), \ s \not\rightsquigarrow_c s'$$

▶ **Definition 13** (Token Machine Rewriting System). *We define a transition system $\rightsquigarrow$ as* exactly *one $\rightsquigarrow_d$ rule followed by all possible $\rightsquigarrow_c$ rules. In other words,*

$$t \rightsquigarrow u \ \textit{iff} \ (\exists t' \cdot t \rightsquigarrow_d t' \rightsquigarrow_c^* u \quad \textit{and} \quad u \textit{ is collision-free})$$

3.2 Strong Normalization and Confluence

The token machine Rewrite System of Definition 13 ensures that the collisions that can happen always happen. The system does not a priori forbid two tokens on the same edge, provided that they have the same direction. However this is something we want to avoid as there is no good intuition behind it: We want to link the token machine to the standard interpretation, which is not possible if two tokens can appear on the same edge.

In this section we show that, under a notion of well-formedness characterizing token uniqueness on each edge, the Token State Rewrite System $(\rightsquigarrow)$ is strongly normalizing and confluent.

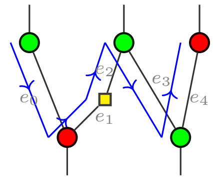
▶ **Definition 14** (Polarity of a Term in a Path). *Let D be a ZX-diagram, and $p \in \mathrm{Paths}(D)$ be a path in D. Let $t = (e, d, x) \in \mathbf{tk}(D)$. Then:*

$$P(p, t) = \begin{cases} 1 & \textit{if } e \in p \textit{ and } e \textit{ is } d\textit{-oriented} \\ -1 & \textit{if } e \in p \textit{ and } e \textit{ is } \neg d\textit{-oriented} \\ 0 & \textit{if } e \notin p \end{cases}$$

We extend the definition to subterms $\alpha\, t_1 ... t_m$ of a token-states t:

$$P(p,0) = P(p,1) = 0, \qquad P(p, \alpha\, t_1 ... t_m) = P(p,t_1) + ... + P(p,t_m).$$

In the following, we shall simply refer to such subterms as "terms of t".



▶ **Example 15.** In the (piece of) diagram presented on the right, the blue directed line $p = (e_0, e_1, e_2, e_3, e_4)$ is a path. The orientation of the edges in the path is represented by the arrow heads, and e_3 for instance is $\downarrow$-oriented in p which implies that we have $P(p,(e_3 \uparrow x)) = -1$.

▶ **Definition 16** (Well-formedness). *Let D be a ZX-diagram, and $s \in \mathbf{tkS}(D)$ a token state on D. We say that s is well-formed if for every term t in s and every path $p \in \mathrm{Paths}(D)$ we have $P(p,t) \in \{-1,0,1\}$.*

▶ **Proposition 17** (Invariance of Well-Formedness). *Well-formedness is preserved by $(\rightsquigarrow)$: if $s \rightsquigarrow^* s'$ and s is well-formed, then s' is well-formed.* ◀

Well-formedness prevents the unwanted scenario of having two tokens on the same wire, and oriented in the same direction (e.g. $(e_0 \downarrow x)(e_0 \downarrow y)$). As shown in the Proposition 18, this property is in fact stronger.

▶ **Proposition 18** (Full Characterisation of Well-Formed Terms). *Let D be a ZX-diagram, and $s \in \mathbf{tkS}(\mathbf{D})$ be* not *well-formed, i.e. there exists a term t in s, and $p \in \mathrm{Paths}(D)$ such that $|P(p,t)| \geq 2$. Then we can rewrite $s \rightsquigarrow s'$ such that a term in s' has a product of at least two tokens of the form $(e_0, d, \_)$.* ◀

Although well-formedness prevents products of tokens on the same wire, it does not guarantee termination: for this we need to consider polarities along cycles.

▶ **Proposition 19** (Invariant on Cycles). *Let D be a ZX-diagram, and $c \in \mathrm{Cycles}(D)$ a cycle. Let $t_1, \ldots, t_n$ be tokens, and s be a token state such that $t_1 ... t_n \rightsquigarrow^* s$. Then for every non-null term t in s we have $P(c, t_1 ... t_n) = P(c,t)$.* ◀

This proposition tells us that the polarity is preserved inside a cycle. By requiring the polarity to be 0, we can show that the token machine terminates. This property is defined formally in the following.

▶ **Definition 20** (Cycle-Balanced Token State). *Let D be a ZX-diagram, and t a term in a token state on D. We say that t is* cycle-balanced *if for all cycles $c \in \mathrm{Cycles}(D)$ we have $P(c,t) = 0$. We say that a token state is* cycle-balanced *if all its terms are cycle-balanced.*

To show that being cycle-balanced implies termination, we need the following intermediate lemma. This essentially captures the fact that a token in the diagram comes from some other token that "traveled" in the diagram earlier on.

▶ **Lemma 21** (Rewinding). *Let D be a ZX-diagram, and t be a term in a well-formed token state on D, and such that $t \rightsquigarrow^* \sum_i \lambda_i t_i$, with $(e_n, d, x) \in t_1$. If t is cycle-balanced, then there exists a path $p = (e_0, ..., e_n) \in \mathrm{Paths}(D)$ such that e_n is d-oriented in p, and $P(p,t) = 1$.* ◀

We can now prove strong-normalization.

▶ **Theorem 22** (Termination of well-formed, cycle-balanced token state). *Let D be a ZX-diagram, and $s \in \mathbf{tkS}(\mathbf{D})$ be well-formed. The token state s is strongly normalizing if and only if it is cycle-balanced.* ◀

Intuitively, this means that tokens inside a cycle will cancel themselves out if the token state is cycle-balanced. Since cycles are the only way to have a non-terminating token machine, we are sure that our machine will always terminate.

▶ **Proposition 23** (Local Confluence). *Let D be a ZX-diagram, and $s \in \mathbf{tkS}(D)$ be well-formed and collision-free. Then, for all $s_1, s_2 \in \mathbf{tkS}(D)$ such that $s_1 \leftsquigarrow s \rightsquigarrow s_2$, there exists $s' \in \mathbf{tkS}(D)$ such that $s_1 \rightsquigarrow^* s' \;{}^*\!\!\leftsquigarrow s_2$.* ◀

▶ **Corollary 24** (Confluence). *Let D be a ZX-diagram. The rewrite system $\rightsquigarrow$ is confluent for well-formed and cycle-balanced token states.* ◀

▶ **Corollary 25** (Uniqueness of Normal Forms). *Let D be a ZX-diagram. A well-formed and cycle-balanced token state admits a unique normal form under the rewrite system $\rightsquigarrow$.* ◀

3.3 Semantics and Structure of Normal Forms

In this section, we discuss the structure of normal forms, and relate the system to the standard interpretation presented in Section 2.

▶ **Proposition 26** (Single-Token Input). *Let $D : n \to m$ be a connected $\mathbf{ZX}$-diagram with $\mathcal{I}(D) = [a_i]_{0<i\leq n}$ and $\mathcal{O}(D) = [b_i]_{0<i\leq m}$, $0 < k \leq n$ and $x \in \{0,1\}$, such that:*

$$[\![D]\!] \circ (id_{k-1} \otimes |x\rangle \otimes id_{n-k}) = \sum_{q=1}^{2^{m+n-1}} \lambda_q \, |y_{1,q}, ..., y_{m,q}\rangle\langle x_{1,q}, ..., x_{k-1,q}, x_{k+1,q}, ..., x_{n,q}|$$

Then:
$$(a_k \downarrow x) \rightsquigarrow^* \sum_{q=1}^{2^{m+n-1}} \lambda_q \prod_i (b_i \downarrow y_{i,q}) \prod_{i\neq k} (a_i \uparrow x_{i,q})$$
◀

This proposition conveys the fact that dropping a single token in state x on wire a_k gives the same semantics as the one obtained from the standard interpretation on the ZX-diagram, with wire a_k connected to the state $|x\rangle$.

Proposition 26 can be made more general. However, we first need the following result on ZX-diagrams:

▶ **Lemma 27** (Universality of Connected ZX-Diagrams). *Let $f : \mathbb{C}^{2^n} \to \mathbb{C}^{2^m}$. There exists a* connected *ZX-diagram $D_f : n \to m$ such that $[\![D_f]\!] = f$.* ◀

▶ **Proposition 28** (Multi-Token Input). *Let D be a* connected *ZX-diagram with $\mathcal{I}(D) = [a_i]_{1\leq i\leq n}$ and $\mathcal{O}(D) = [b_i]_{1\leq i\leq m}$; with $n \geq 1$.*

If:
$$[\![D]\!] \circ \left(\sum_{q=1}^{2^n} \lambda_q \, |x_{1,q}, ..., x_{n,q}\rangle\right) = \sum_{q=1}^{2^m} \lambda'_q \, |y_{1,q}, ..., y_{m,q}\rangle$$

then:
$$\sum_{q=1}^{2^n} \lambda_q \prod_{i=1}^{n} (a_i \downarrow x_{i,q}) \rightsquigarrow^* \sum_{q=1}^{2^m} \lambda'_q \prod_{i=1}^{m} (b_i \downarrow y_{i,q})$$
◀

This proposition is a direct generalization of the proposition 26. Thanks to all of that, we can show that we can start evaluating not only on a single or even multiple input wires, but in fact on any wire in the ZX-diagram, as long as we respect well-formedness and cycle-balancedness. But we need to be careful about collisions. For that to hold, we need to rewrite each part of the sum independently before computing the sum.

▶ **Theorem 29** (Arbitrary Wire Initialisation). *Let D be a connected ZX-diagram, with $\mathcal{I}(D) = [a_i]_{1\leq i\leq n}$, $\mathcal{O}(D) = [b_i]_{1\leq i\leq m}$, and $e \in \mathcal{E}(D) \neq \emptyset$ such that $(e \downarrow x)(e \uparrow x) \rightsquigarrow^* t_x$ for $x \in \{0,1\}$*

with t_x terminal (the rewriting terminate by Corollary 25). Then:

$$[\![D]\!] = \sum_{q=1}^{2^{m+n}} \lambda_q \, |y_{1,q} \dots y_{m,q}\rangle\langle x_{1,q} \dots x_{n,q}| \implies t_0 + t_1 = \sum_{q=1}^{2^{m+n}} \lambda_q \prod_i (b_i {\downarrow} y_{i,q}) \prod_i (a_i {\uparrow} x_{i,q}) \qquad \blacktriangleleft$$

4 Sum-Over-Paths Token Machine

A serious drawback of the previous token machine is that the token state grows exponentially quickly in the number of nodes in the diagram. A more compact representation (linear in the size of the diagram as we will see in Prop. 36) can be obtained by adapting the concept of sums-over-paths (SOP) [1] to our machine. This can be obtained naturally, as strong links between ZX-Calculus and SOP morphisms were already shown to exist [33, 40]. Intuitively, SOP will allow us to manipulate token states in a symbolic way, where for instance $(e \downarrow 0) + (e \downarrow 1)$ will be represented by $(e \downarrow y)$.

▶ **Definition 30.** *Let D be a ZX-diagram. A* **SOP***-token is a triplet (p, d, B) belonging to $\mathcal{E}(\mathcal{D}) \times \{\downarrow, \uparrow\} \times \mathbb{F}_2[\vec{y}]$ where $\vec{y} := (y_i)_{0 \le i < n}$ are n variables from a set of variables $\mathcal{V}$; and where $\mathbb{F}_2 := \mathbb{Z}/2\mathbb{Z}$ is the Galois field of order 2. We denote the set of* **SOP***-tokens on D with variables $\vec{y}$ by $\mathbf{tk_{SOP}}(D)[\vec{y}]$. A* **SOP***-token-state is a quadruplet:*

$$(s, \vec{y}, P, \{t_j\}_{0 \le j < p}) \in \mathbb{R} \times \mathcal{V}^n \times \mathbb{R}[\vec{y}]/(1, \{y_i^2 - y_i\}_{0 \le i < n}) \times \mathbf{tk_{SOP}}(D)[\vec{y}]^p$$

where $\mathbb{R}[\vec{y}]/(1, \{y_i^2 - y_i\}_{0 \le i < n})$ is the set of real-valued multivariate polynomials (whose variables are $\vec{y}$), modulo 1 and modulo $(y_i^2 - y_i)$ for all variables y_i. For any valuation of $\vec{y}$, $2\pi P(\vec{y})$ represents an angle, hence P is taken modulo 1. Since each y_i is a boolean variable, we can consider $y_i^2 - y_i = 0$. To better reflect what this quadruplet represents, we usually write it as:

$$s \sum_{\vec{y}} e^{2i\pi P(\vec{y})} (p_0, d_0, B_0(\vec{y}))...(p_{m-1}, d_{m-1}, B_{m-1}(\vec{y}))$$

We denote the set of **SOP***-token-states on D by $\mathbf{tkS_{SOP}}(D)$.*

▶ **Example 31.** Let $D =$ 

, then $\frac{1}{\sqrt{2}} \sum_{y_0, y_1} e^{2i\pi \frac{y_0 y_1}{2}} (e_0 \uparrow y_0)(e_1 \downarrow y_1) \in \mathbf{tkS_{SOP}}(D)$.

We can link this formalism back to the previous one, by defining a map that associates any **SOP**-token-state to a "usual" token-state. This map simply evaluates the term by having all its variables span $\{0, 1\}$:

▶ **Definition 32.** *We define $[.]^{\mathbf{tk}} : \mathbf{tkS_{SOP}}(D) \to \mathbf{tkS}(D)$ by:*

$$\left[s \sum_{\vec{y}} e^{2i\pi P(\vec{y})} \prod_j (p_j, d_j, B_j(\vec{y})) \right]^{\mathbf{tk}} := s \sum_{\vec{y} \in \{0,1\}^n} e^{2i\pi P(\vec{y})} \prod_j (p_j, d_j, B_j(\vec{y}))$$

▶ **Example 33.**

$$\left[\frac{1}{\sqrt{2}} \sum_{y_0, y_1} e^{2i\pi \frac{y_0 y_1}{2}} (e_0 \uparrow y_0)(e_1 \downarrow y_1) \right]^{\mathbf{tk}} = \frac{1}{\sqrt{2}} \begin{pmatrix} (e_0 \uparrow 0)(e_1 \downarrow 0) + (e_0 \uparrow 1)(e_1 \downarrow 0) \\ +(e_0 \uparrow 0)(e_1 \downarrow 1) - (e_0 \uparrow 1)(e_1 \downarrow 1) \end{pmatrix}$$

We give the adapted set of rewrite rules for our **SOP**-token-machine in Table 2. In the rewrite rules of our token machine, we have to map elements of $\mathbb{F}_2[\vec{y}]$ to elements of $\mathbb{R}[\vec{y}]/(1, \{y_i^2 - y_i\})$ for the Boolean polynomials to be sent to the phase polynomial. The map $\widehat{(.)} : \mathbb{F}_2[\vec{y}] \to \mathbb{R}[\vec{y}]/1(1, \{y_i^2 - y_i\})$ that does this is defined as:

$$\widehat{B \oplus B'} = \widehat{B} + \widehat{B'} - 2\widehat{B}\widehat{B'} \qquad \widehat{BB'} = \widehat{B}\widehat{B'} \qquad \widehat{y_i} = y_i \qquad \widehat{0} = 0 \qquad \widehat{1} = 1$$

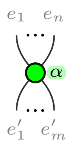
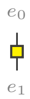
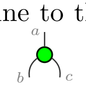
	$(e_0 \downarrow B)(e_0 \uparrow B') \rightsquigarrow_c \frac{1}{2}\sum_z e^{2i\pi \frac{z}{2}(\widehat{B \oplus B'})}$	(Collision)
	$(e_b \downarrow B) \rightsquigarrow_d (e_{\neg b} \uparrow B)$	(◡-diffusion)
	$(e_b \uparrow B) \rightsquigarrow_d (e_{\neg b} \downarrow B)$	(◠ -diffusion)
	$(e_k \downarrow B) \rightsquigarrow_d e^{2i\pi(\frac{\alpha}{2\pi}\widehat{B})} \prod_{j \neq k}(e_j \uparrow B) \prod_j (e'_j \downarrow B)$ $(e'_k \uparrow B) \rightsquigarrow_d e^{2i\pi(\frac{\alpha}{2\pi}\widehat{B})} \prod_j (e_j \uparrow B) \prod_{j \neq k}(e'_j \downarrow B)$	(-Diffusion)
	$(e_0 \downarrow B) \rightsquigarrow_d \frac{1}{\sqrt{2}}\sum_z e^{2i\pi(\frac{z}{2}\widehat{B})}(e_1 \downarrow z)$ $(e_1 \uparrow B) \rightsquigarrow_d \frac{1}{\sqrt{2}}\sum_z e^{2i\pi(\frac{z}{2}\widehat{B})}(e_0 \uparrow z)$	(-Diffusion)

Table 2 Rewrite rules for $\rightsquigarrow_{\mathbf{sop}}$.

The provided rewrite rules do not give the full picture, for simplicity. If a rule gives $(e, d, b) \rightsquigarrow_{\mathbf{sop}} s' \sum_{\vec{y}'} e^{2i\pi P'} \prod_j (e'_j, d'_j, b'_j)$, we have to apply it to a full **SOP**-token-state as follows: $s \sum_{\vec{y}} e^{2i\pi P}(e, d, b) \prod_j (e_j, d_j, b_j) \rightsquigarrow ss' \sum_{\vec{y},\vec{y}'} e^{2i\pi(P+P')} \prod_j (e'_j, d'_j, b'_j) \prod_j (e_j, d_j, b_j)$. Just as before, the rewrite system is defined by first applying a diffusion rule then all possible collision rules.

This set of rules mimics the previous one for **SOP**-token-states, except that it "synchronizes" rewrites on all the terms at once (but not on all tokens).

▶ **Example 34.** Let us compare the behavior of the previous token machine to the SOP machine. We send tokens in states 0 and 1 down the wire a in the diagram  . In the former machine, this leads to

$$(a \downarrow 0) + (a \downarrow 1) \rightsquigarrow (b \downarrow 0)(c \downarrow 0) + (a \downarrow 1) \rightsquigarrow (b \downarrow 0)(c \downarrow 0) + (b \downarrow 1)(c \downarrow 1).$$

while in the latter: $\sum_y (a \downarrow y) \rightsquigarrow_{\mathbf{sop}} \sum_y (b \downarrow y)(c \downarrow y)$

In both cases the result is the same when interpreted as usual token states. We notice that the $\rightsquigarrow_{\mathbf{sop}}$ token machine only took one step compared to the standard one, which leads to the following proposition:

▶ **Proposition 35.** *For any $D \in \mathbf{ZX}$ and $s, s' \in \mathbf{tkS_{SOP}}(D)$, whenever $s \rightsquigarrow_{\mathbf{sop}} s'$ we have $[s]^{\mathbf{tk}} \rightsquigarrow^* [s']^{\mathbf{tk}}$.* ◀

We can show a result on the growth size of the token-state as it rewrites, which was the motivation for the use of this formalism.

▶ **Proposition 36.** *Let $D \in \mathbf{ZX}$ and $s, s' \in \mathbf{tkS_{SOP}}(D)$ such that all Boolean polynomials B_j in s are reduced to a single term of degree ≤ 1, and such that $s \rightsquigarrow_{\mathbf{sop}} s'$. Then, the size of s' is bounded by: $\mathcal{S}(s') \leq \mathcal{S}(s) + \Delta(D)$ where $\mathcal{S}$ denotes the cumulative number of terms in the phase polynomial and the number of tokens in the token-state, and where $\Delta(D)$ represents the maximum arity of generators in D.* ◀

The requirement on Boolean polynomials may seem overly restrictive. However, it is invariant under rewriting: starting with a token-state in this form ensures polynomial growth.

Polarity can be defined in this setting (and is even more natural, as we do not need to consider each term individually) providing the notions of well-formedness and cycle-balancedness. The main results from Section 4 are valid in this setting. We recover strong normalization for well-formed, cycle-balanced token-states (Theorem 22), Local Confluence (Proposition 23) and their corollaries, such as uniqueness of normal forms (Corollary 25).

Non-empty terminal token states can also be interpreted as SOP-morphisms. Suppose an SOP-token state $S = s\sum_{\vec{y}'} e^{2i\pi P} \prod_i (b_i \downarrow B_i(\vec{y}')) \prod_i (a_i \uparrow A_i(\vec{y}'))$ on a diagram D with $\mathcal{I}(\mathcal{D}) = [a_i]_{1\leq i\leq n}$ and $\mathcal{O}(\mathcal{D}) = [b_i]_{1\leq i\leq m}$. Then $[S]^{\mathbf{SOP}} := s\sum_{\vec{y}} e^{2i\pi P(\vec{y})} \left|B_0(\vec{y}),...\right\rangle\!\left\langle A_0(\vec{y}),...\right|$ is the SOP morphism associated to S. We have the following commutative diagram:

$$\begin{tikzcd} \mathbf{tkS_{SOP}}\downarrow \arrow[r, "{[.]^{\mathbf{tk}}}"] \arrow[d, "{[.]^{\mathbf{SOP}}}"] & \mathbf{tkS}\downarrow \arrow[d, "{[\![.]\!]}"] \\ \mathbf{SOP} \arrow[r, "{[\![.]\!]}"] & \mathbf{Qubit} \end{tikzcd}$$

where $\mathbf{tkS_{SOP}}\downarrow$ (resp. $\mathbf{tkS}\downarrow$) is the set of non-empty well-formed terminal SOP-token states (resp. token states), and $\mathbf{tkS}\downarrow \xrightarrow{[\![.]\!]} \mathbf{Qubit}$ is the interpretation obtained from Theorem 29.

5 Extension to Mixed Processes

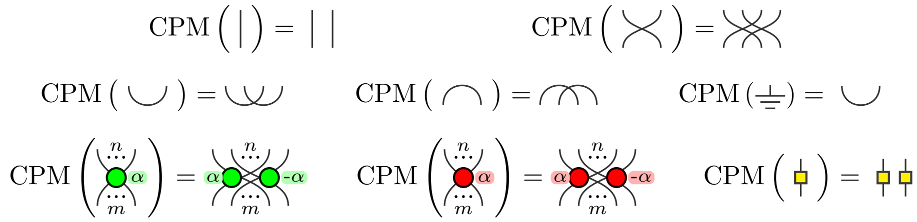
The token machines presented so far worked for so-called *pure* quantum processes i.e. with no interaction with the environment. To demonstrate how generic our approach is, we show how to adapt it to the natural extension of *mixed* processes, represented with completely positive maps (CPM). This in particular allows us to represent quantum measurements.

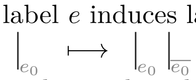
5.1 ZX-diagrams for Mixed Processes

The interaction with the environment can be modeled in the ZX-Calculus by adding a unary generator ⏚ to the language [8, 5], intuitively enforcing the state of the wire to be classical. We denote with $\mathbf{ZX}^{⏚}$ the set of diagrams obtained by adding ⏚ to the usual generators of the ZX-Calculus.

Similar to what is done in quantum computation, the standard interpretation $[\![.]\!]^{⏚}$ for $\mathbf{ZX}^{⏚}$ maps diagrams to CPMs. If $D \in \mathbf{ZX}$ we define $[\![D]\!]^{⏚}$ as $\rho \mapsto [\![D]\!]^{\dagger} \circ \rho \circ [\![D]\!]$, and we set $[\![⏚]\!]^{⏚}$ as $\rho \mapsto \mathrm{Tr}(\rho)$, where $\mathrm{Tr}(\rho)$ is the trace of ρ.

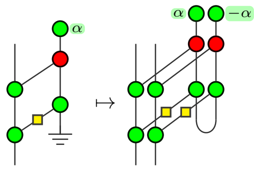
There is a canonical way to map a $\mathbf{ZX}^{⏚}$-diagram to a $\mathbf{ZX}$-diagram in a way that preserves the semantics: the so-called CPM-construction [37]. We define the map (conveniently named) CPM as the map that preserves compositions $(\_ \circ \_)$ and $(\_ \otimes \_)$ and such that:



With respect to what happens to edge labels, notice that every edge in D can be mapped to 2 edges in CPM(D). We propose that label e induces label e in the first copy, and $\overline{e}$ in the second, e.g, for the identity diagram: 

In the general ZX-Calculus, it has been shown that the axiomatization itself could be extended to a complete one by adding only 4 axioms [5].

▶ **Example 37.** A $\mathbf{ZX}^{\doteqdot}$-diagram and its associated CPM construction is shown on the right.



5.2 Token Machine for Mixed Processes

We now aim to adapt the token machine to $\mathbf{ZX}^{\doteqdot}$, the formalism for completely positive maps. Since the formalism of sum-over-paths gave us an easier machine to work with, where terms are smaller while guaranteeing a simulation result with respect to the first token machine, we will use it to define the token machine for completely positive maps.

▶ **Definition 38.** *Let D be a ZX-diagram. A $\mathbf{SOP}^{\doteqdot}$-token is a quadruplet $(p, d, B, B') \in \mathcal{E}(D) \times \{\downarrow, \uparrow\} \times \mathbb{F}_2[\vec{y}] \times \mathbb{F}_2[\vec{y}]$ where $\vec{y} := (y_i)_{0 \leq i < n}$ are variables from a set of variables $\mathcal{V}$. We denote the set of $\mathbf{SOP}^{\doteqdot}$-tokens on D with variables $\vec{y}$ by $\mathbf{tk}^{\doteqdot}_{\mathbf{SOP}}(D)[\vec{y}]$. Similar to what was done in Definition 30, a $\mathbf{SOP}^{\doteqdot}$-token-state is a quadruplet*

$$(s, \vec{y}, P, \{t_j\}_{0 \leq j < p}) \in \mathbb{R} \times \mathcal{V}^n \times \mathbb{R}[\vec{y}]/(1, \{y_i^2 - y_i\}_{0 \leq i < n}) \times \mathbf{tk}^{\doteqdot}_{\mathbf{SOP}}(D)[\vec{y}]^p$$

To better reflect what this quadruplet represents, we usually write it as:

$$s \sum_{\vec{y}} e^{2i\pi P(\vec{y})}(p_0, d_0, B_0(\vec{y}), B'_0(\vec{y}))\ldots(p_{m-1}, d_{m-1}, B_{m-1}(\vec{y}), B'_{m-1}(\vec{y}))$$

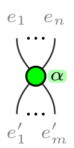
We denote the set of $\mathbf{SOP}^{\doteqdot}$-token-states on D by $\mathbf{tkS}^{\doteqdot}_{\mathbf{SOP}}(D)$

In other words, the difference with the previous machine is that tokens here have an additional Boolean function (e.g. $(a \downarrow x, y)$). The rewrite rules are given in Table 3.

$$\begin{matrix}(e_0 \downarrow B_0, B_1) \\ \times \\ (e_0 \uparrow B'_0, B'_1)\end{matrix} \rightsquigarrow_c \frac{1}{4} \sum_{z_0, z_1} e^{2i\pi(\frac{z_0}{2}\widehat{B_0 \oplus B_1} + \frac{z_1}{2}\widehat{B'_0 \oplus B'_1})} \qquad \text{(Collision)}$$

$$(e_b \downarrow B, B') \rightsquigarrow_d (e_{\neg b} \uparrow B, B') \qquad (\smile\text{-diffusion})$$

$$(e_b \uparrow B, B') \rightsquigarrow_d (e_{\neg b} \downarrow B, B') \qquad (\frown\text{-diffusion})$$



$$(e_k \downarrow B_0, B_1) \rightsquigarrow_d e^{2i\pi\frac{\alpha}{2\pi}(\widehat{B_0} - \widehat{B_1})} \begin{matrix}\prod_{j \neq k}(e_j \uparrow B_0, B_1) \\ \times \\ \prod_j(e'_j \downarrow B_0, B_1)\end{matrix}$$

$$(e'_k \uparrow B_0, B_1) \rightsquigarrow_d e^{2i\pi\frac{\alpha}{2\pi}(\widehat{B_0} - \widehat{B_1})} \begin{matrix}\prod_j(e_j \uparrow B_0, B_1) \\ \times \\ \prod_{j \neq k}(e'_j \downarrow B_0, B_1)\end{matrix} \qquad \text{(-Diffusion)}$$

$$(e_0 \downarrow B, B') \rightsquigarrow_d \frac{1}{2} \sum_{z, z'} e^{2i\pi(\frac{z}{2}\widehat{B} + \frac{z'}{2}\widehat{B'})}(e_1 \downarrow z, z')$$

$$(e_1 \uparrow B, B') \rightsquigarrow_d \frac{1}{2} \sum_{z, z'} e^{2i\pi(\frac{z}{2}\widehat{B} + \frac{z'}{2}\widehat{B'})}(e_0 \uparrow z, z') \qquad \text{(-Diffusion)}$$

$$(e_0 \downarrow B, B') \rightsquigarrow_d \frac{1}{2} \sum_{z} e^{2i\pi\frac{z}{2}(\widehat{B \oplus B'})} \qquad \text{(Trace-Out)}$$

Table 3 The rewrite rules for $\rightsquigarrow_{\doteqdot}$.

It is possible to link this formalism back to the mixed processes-free **SOP**-token-states, using the existing CPM construction for ZX-diagrams. We extend this map by CPM : $\mathbf{tkS}^{\doteq}_{\mathbf{SOP}}(D) \to \mathbf{tkS}_{\mathbf{SOP}}(\mathrm{CPM}(D))$, defined as:

$$s\sum_{\vec{y}} e^{2i\pi P(\vec{y})} \prod_j (p_j, d_j, B_j(\vec{y}), B'_j(\vec{y})) \mapsto s\sum_{\vec{y}} e^{2i\pi P(\vec{y})} \prod_j (p_j, d_j, B_j(\vec{y}))(\overline{p_j}, d_j, B'_j(\vec{y}))$$

$\mathrm{CPM}(D)$ can be seen as two copies of D where $\doteq$ is replaced. Each token in D corresponds to two tokens in $\mathrm{CPM}(D)$, at the same spot but in the two copies of D. The two Boolean polynomials B and B' represent the Boolean polynomials of the two corresponding tokens.

We can then show that this rewriting system is consistent:

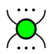
▶ **Theorem 39.** *Let D be a $\mathbf{ZX}^{\doteq}$-diagram, and $t_1, t_2 \in \mathbf{tkS}^{\doteq}_{\mathbf{SOP}}(D)$. Then whenever $t_1 \rightsquigarrow_{\doteq} t_2$ we have $\mathrm{CPM}(t_1) \rightsquigarrow^{\{1,2\}}_{\mathbf{sop}} \mathrm{CPM}(t_2)$.* ◀

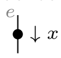
In fact, the $\rightsquigarrow_{\doteq}$ rewriting rule will only be simulated by 2 rewriting rules ($\rightsquigarrow_{\mathbf{sop}}$), except in the case of the Trace-out where ($\rightsquigarrow_{\mathbf{sop}}$) only needs to apply one rule.

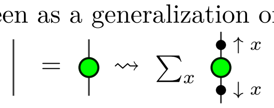
Again, the notions of polarity, well-formedness and cycle-balancedness can be adapted, and again, we get strong normalization(Theorem 22), confluence (Corollary 24), and uniqueness of normal forms (Corollary 25) for well-formed and cycle-balanced token states.

6 Conclusion and Future Work

Since quantum circuits can be mapped to ZX-diagrams, our token machines induce a notion of asynchronicity for quantum circuits. This contrasts with the notion of token machine defined in [32] where some form of synchronicity is enforced.

Our token machine can however be made synchronous: all tokens in a token state then move at once. This implies adapting the rules to take into account all incoming tokens for each generator. For instance, in the [ -Diffusion]-rule the product $\prod_i (e_i \downarrow x_i)$ rewrites into $\delta_{x_1,\ldots,x_n} e^{i\alpha x_1} \prod_i (e'_i \downarrow x_1)$. This notion of synchronicity is to be contrasted with [32] where tokens have to wait for all other incoming tokens before going through a gate.

The presentation we followed clearly distinguishes between ZX-diagrams and token states on them. We could instead see tokens as part of the ZX-diagram. For instance, $(e \downarrow x)$ on D could be a literal node  on D. For our first token machine, this would imply representing a token state by a sum of diagrams with tokens on them. In the SOP framework, however, we would simply get a single diagram with tokens on them and global scalar and polynomial in the variables.

In this paper, we showed that our tokens could start at any edge, in a configuration that respects well-formedness and cycle-balancedness. We may also consider a “pulse” version, in which each node emits one token in all of its edges at once, during the evaluation of the token machine. This pulse version can be seen as a generalization of the initialization of the token state in Theorem 29: the intuition is  .

References

1 Matthew Amy. Towards large-scale functional verification of universal quantum circuits. In Peter Selinger and Giulio Chiribella, editors, *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Canada, 3-7th June 2018*, volume 287 of *EPTCS*, pages 1–21, 2018. doi:10.4204/EPTCS.287.1.

2 Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the λ-calculus. *Theoretical Computer Science*, 142(2):277–297, 1995.

3 Miriam Backens. The ZX-calculus is complete for stabilizer quantum mechanics. *New Journal of Physics*, 16(9):093021, 2014. `doi:10.1088/1367-2630/16/9/093021`.

4 Miriam Backens, Hector Miller-Bakewell, Giovanni de Felice, Leo Lobski, and John van de Wetering. There and back again: A circuit extraction tale, 2020. `arXiv:2003.01664`.

5 Titouan Carette, Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Completeness of Graphical Languages for Mixed States Quantum Mechanics. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 108:1–108:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2019/10684`, `doi:10.4230/LIPIcs.ICALP.2019.108`.

6 Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. A deductive verification framework for circuit-building quantum programs. `arXiv:2003.05841`. To appear in *Proceedings of ESOP'21*.

7 Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.

8 Bob Coecke and Simon Perdrix. Environment and Classical Channels in Categorical Quantum Mechanics. *Logical Methods in Computer Science*, Volume 8, Issue 4, Nov 2012. URL: `https://lmcs.episciences.org/719`, `doi:10.2168/LMCS-8(4:14)2012`.

9 Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal λ-machines. *Theoretical Computer Science*, 227(1-2):79–97, 1999.

10 Niel de Beaudrap, Ross Duncan, Dominic Horsman, and Simon Perdrix. Pauli Fusion: a computational model to realise quantum transformations from ZX terms. In *QPL'19 : International Conference on Quantum Physics and Logic*, Los Angeles, United States, June 2019. 12 pages + appendices. URL: `https://hal.archives-ouvertes.fr/hal-02413388`.

11 Niel de Beaudrap and Dominic Horsman. The ZX calculus is a language for surface code lattice surgery. *Quantum*, 4:218, January 2020. `doi:10.22331/q-2020-01-09-218`.

12 Ross Duncan and Liam Garvie. Verifying the smallest interesting colour code with quantomatic. In Bob Coecke and Aleks Kissinger, editors, *Proceedings 14th International Conference on Quantum Physics and Logic, Nijmegen, The Netherlands, 3-7 July 2017*, volume 266 of *Electronic Proceedings in Theoretical Computer Science*, pages 147–163, 2018. `doi:10.4204/EPTCS.266.10`.

13 Ross Duncan, Aleks Kissinger, Simon Perdrix, and John Van De Wetering. Graph-theoretic simplification of quantum circuits with the zx-calculus. *Quantum*, 4:279, 2020.

14 Ross Duncan and Maxime Lucas. Verifying the Steane code with Quantomatic. In Bob Coecke and Matty Hoban, editors, *Proceedings of the 10th International Workshop on Quantum Physics and Logic, Castelldefels (Barcelona), Spain, 17th to 19th July 2013*, volume 171 of *Electronic Proceedings in Theoretical Computer Science*, pages 33–49. Open Publishing Association, 2014. `doi:10.4204/EPTCS.171.4`.

15 Ross Duncan and Simon Perdrix. Rewriting measurement-based quantum computations with generalised flow. *Lecture Notes in Computer Science*, 6199:285–296, 2010. `doi:10.1007/978-3-642-14162-1_24`.

16 Elizabeth Gibney. Quantum gold rush: the private funding pouring into quantum start-ups. *Nature*, 574:22–24, 2019. `doi:10.1038/d41586-019-02935-4`.

17 Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

18 Jean-Yves Girard. Geometry of interaction II: deadlock-free algorithms. In *International Conference on Computer Logic*, pages 76–93. Springer, 1988.

19 Jean-Yves Girard. Geometry of interaction I: interpretation of system f. In *Studies in Logic and the Foundations of Mathematics*, volume 127, pages 221–260. Elsevier, 1989.

20 Jean-Yves Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92(69-108):6, 1989.

21 Jean-Yves Girard. Geometry of interaction III: accommodating the additives. *London Mathematical Society Lecture Note Series*, pages 329–389, 1995.

22 Jean-Yves Girard. Proof-nets: the parallel syntax for proof-theory. *Lecture Notes in Pure and Applied Mathematics*, pages 97–124, 1996.

23 Jean-Yves Girard. Geometry of interaction IV: the feedback equation. In *Logic Colloquium*, volume 3, pages 76–117, 2006.

24 Jean-Yves Girard. Geometry of interaction V: logic in the hyperfinite factor. *Theoretical Computer Science*, 412(20):1860–1883, 2011.

25 Jean-Yves Girard. Geometry of interaction VI: a blueprint for transcendental syntax. *preprint*, 2013.

26 Amar Hadzihasanovic, Kang Feng Ng, and Quanlong Wang. Two complete axiomatisations of pure-state qubit quantum computing. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 502–511, New York, NY, USA, 2018. ACM. URL: `http://doi.acm.org/10.1145/3209108.3209128`, `doi:10.1145/3209108.3209128`.

27 Anne Hillebrand. Quantum protocols involving multiparticle entanglement and their representations. Master's thesis, University of Oxford, 2011. URL: `https://www.cs.ox.ac.uk/people/bob.coecke/Anne.pdf`.

28 Lars Jaeger. *The Second Quantum Revolution*. Springer, 2018.

29 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A complete axiomatisation of the ZX-calculus for Clifford+T quantum mechanics. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 559–568, New York, NY, USA, 2018. ACM. URL: `http://doi.acm.org/10.1145/3209108.3209131`, `doi:10.1145/3209108.3209131`.

30 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Diagrammatic reasoning beyond Clifford+T quantum mechanics. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 569–578, New York, NY, USA, 2018. ACM. URL: `http://doi.acm.org/10.1145/3209108.3209139`, `doi:10.1145/3209108.3209139`.

31 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A generic normal form for zx-diagrams and application to the rational angle completeness. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10, June 2019. `doi:10.1109/LICS.2019.8785754`.

32 Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The geometry of parallelism. classical, probabilistic, and quantum effects. *CoRR*, abs/1610.09629, 2016. URL: `http://arxiv.org/abs/1610.09629`, `arXiv:1610.09629`.

33 Louis Lemonnier, John van de Wetering, and Aleks Kissinger. Hypergraph simplification: Linking the path-sum approach to the zh-calculus, 2020. arXiv:2003.13564. `arXiv:2003.13564`.

34 Alexandre Ménard, Ivan Ostojic, Mark Patel, , and Daniel Volz. A game plan for quantum computing. *McKinsey Quaterly*, February 2020.

35 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2002.

36 Qureca.com. Overview on quantum initiatives worldwide. `https://www.qureca.com/overview-on-quantum-initiatives-worldwide/`, January 2021.

37 Peter Selinger. Dagger compact closed categories and completely positive maps. *Electronic Notes in Theoretical computer science*, 170:139–163, 2007.

38 Renaud Vilmart. A near-minimal axiomatisation of zx-calculus for pure qubit quantum mechanics. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10, June 2019. `doi:10.1109/LICS.2019.8785765`.

39 Renaud Vilmart. *ZX-Calculi for Quantum Computing and their Completeness*. Theses, Université de Lorraine, September 2019. URL: `https://hal.archives-ouvertes.fr/tel-02395443`.

40 Renaud Vilmart. The Structure of Sum-Over-Paths, its Consequences, and Completeness for Clifford, March 2020. arXiv:2003.05678. `arXiv:2003.05678`.

A Proofs of Section 3

Proof of Proposition 17. Let D be a ZX-diagram, and s be a well-formed token state on D. Let t be a term of s, and e_0 be the edge where a rewriting occurs. If the rewriting does not affect t, then the well-formedness of t obviously holds. If it does, and $t \rightsquigarrow_{c,d} \sum_q t_q$, we have to check two cases:

- Collision: let $p \in \text{Paths}(D)$. If no tokens remain in the term t_q, then $P(p, t_q) = 0$. Otherwise:
  - if $e_0 \notin p$, then $P(p, t_q) = P(p, t)$
  - if $e_0 \in p$, then $P(p, t_q) = P(p, t) + 1 - 1$ because the two tokens have alternating polarity
- Diffusion: let $p \in \text{Paths}(D)$, and $(e_0, d, x) \rightsquigarrow_d \sum_q \lambda_q \prod_{i \in S}(e_i, d_i, x_{i,q})$ (this captures all possible diffusion rules).
  - if $e_0 \notin p$ and $\forall i, e_i \notin p$, then $P(p, t_q) = P(p, t)$
  - if $e_0 \in p$ and $\exists k \in S, e_k \in p$, then $\forall i \neq k, e_i \notin p$, because the generator can only be passed through once by the path p. We have $P(p, (e_0, d, x)) = P(p, (e_k, d_k, x_{k,q})$ by the definition of orientation in a path, which means that $\forall q, P(p, t_q) = P(p, t)$
  - if $e_0 \in p$ and $\forall i, e_i \notin p$, then, either *i)* p ends with e_0 and e_0 is d-oriented in p, or *ii)* p starts with e_0 and e_0 is $\neg d$-oriented in p. In both cases, since that $p \setminus \{e_0\}$ is still a path, we have $P(p \setminus \{e_0\}, t) \in \{-1, 0, 1\}$ and since $P(p, t_q) = P(p \setminus \{e_0\}, t)$, we deduce that t_q is still well-formed
  - if $e_0 \notin p$ but $\exists k \in S, e_k \in p$, either e_k is an extremity of p, or $\exists k', e_{k'} \in p$. In the latter case, the tokens in e_k and $e_{k'}$ will have alternating polarity in p, so $\forall q, P(p, t_q) = P(p, t) + 1 - 1$. In the first case, we can show in a way similar to the previous point, that $P(p, t_q) = P(p \setminus \{e_k\}, t) \in \{-1, 0, 1\}$

◀

Proof of Proposition 18. Let t be a term in s, and $p = (e_0, ..., e_n)$ such that $P(p, t) \geq 2$. We can show that we can rewrite t into a token state with term $t' = (e_i, d, \_)(e_i, d, \_)t''$. We do so by induction on $n = |p| - 1$.

If $n = 0$, we have a path constituted of one edge, such that $|P(p, t)| \geq 2$. Even after doing all possible collisions, we are left with $|P(p, t)|$ tokens on e_0, and oriented accordingly.

For $n + 1$, we look at e_0, build $p' := (e_1, ..., e_n)$, and distinguish four cases. If there is no token on e_0, we have $P(p', t) = P(p, t)$, so the result is true by induction hypothesis on p'. If we have a product of at least two tokens going in the same direction, the result is directly true. If we have exactly one token going in each direction, we apply the collision rules, and still have $P(p', t) = P(p, t)$, so the result is true by induction hypothesis on p'. Finally, if we have exactly one token $(e_0, d, \_)$ on e_0, either e_0 is not d-oriented, in which case $P(p', t) = P(p, t) + 1$, or e_0 is d-oriented, in which case the adequate diffusion rule on $(e_0, d, \_)$ will rewrite $t \rightsquigarrow \sum_q t_q$ with $P(p', t_q) = P(p, t)$. ◀

Proof of Proposition 19. The proof can be adapted from the previous one, by forgetting the cases related to the extremity of the paths, as well as the null terms (which can arise from collisions). It can then be observed that the quantity P in this simplified setting is more than bounded to $\{-1,0,1\}$, but preserved. ◀

Proof of Lemma 21. We reason by induction on the length k of the rewrite that leads from t to $\sum_i \lambda_i t_i$.

If $k=0$, we have $(e_n,d,x)\in t$, so the path $p:=(e_n)$ is sufficient.

For $k+1$, suppose $t \rightsquigarrow \sum_i \lambda_i t_i$, and $t_1 \rightsquigarrow^k \sum_j \lambda'_j t'_j$ (hence $t \rightsquigarrow^{k+1} \sum_{i\neq 1}\lambda_i t_i + \sum_j \lambda'_j t'_j$), with $(e_n,d,x)\in t'_1$. By induction hypothesis, there is $p=(e_0,...,e_n)$ such that $P(p,t_1)=1$. We now need to look at the first rewrite from t.

- if the rewrite concerns a generator not in p, then $P(p,t)=P(p,t_1)=1$
- if the rewrite is a collision, then $P(p,t)=P(p,t_1)=1$
- if the rewrite is $(e,d_e,x_e)\rightsquigarrow \sum_q \lambda_q \prod_i (e'_i,d_i,x_{i,q})$
  - if $e\in p$ and $e'_1\in p$, then $P(p,t)=P(p,t_1)=1$
  - if $e'_1\in p$ and $e'_2\in p$, then $P(p,t)=P(p,t_1)-1+1=1$
  - the case $e\in p$ and $\forall i,\ e'_i\notin p$ is impossible:
    * if e is not d_e-oriented in p, it means $e=e_0$, hence $P((e_1,...,e_n),t)=P(p,t)+1=2$ which is forbidden by well-formedness
    * if e is d_e-oriented in p, it means $e=e_n$, which would imply that $P(p,t_1)=0$
  - if $e\notin p$ and $e'_1\in p$ and $\forall i\neq 1, e'_i\notin p$, then $P(e::p,t)=P(p,t_1)=1$, since well-formedness prevents the otherwise possible situation $P(e::p,t)=P(p,t_1)+1=2$. However, $e::p$ may not be a path anymore. If $c=(e,e_0,...,e_\ell)$ forms a cycle, then, since $P(c,t)=0$, we can simply keep the path $p':=(e_{\ell+1},...,e_n)$ with $P(p',t)=1$

◀

Proof of Theorem 22. [$\Rightarrow$]: Suppose $\exists c\in \mathrm{Cycles}(D)$ and t a term of s such that $P(c,t)\neq 0$. By well-formedness, $P(c,t)\in\{-1,1\}$. Any terminal term t' has $P(c,t')=0$, so by preservation of the quantity $P(c,\_)$, t (and henceforth s) cannot terminate.

[$\Leftarrow$]: We are going to show for the reciprocal that, if t is well-formed, and if the constraint $P(c,t)=0$ is verified for every cycle c, then any generator in the diagram can be visited at most once. More precisely, we show that if a generator is visited in a term t, then it cannot be visited anymore in all the terms derived from t. However, the same generator can be visited once for each superposed term (e.g. once in t_1 and once in t_2 for the token state t_1+t_2).

Consider an edge e with token exiting generator g in the term t. Suppose, by reductio ad absurdum, that a token will visit g again in t' (obtained from t), by edge e_n with orientation d. By Lemma 21, there exists a path $p=(e_0,...,e_n)$ such that $P(p,t)=1$ and e_n is d-oriented. Since $e\notin p$ (we would not have a path then), then $p':=(e_0,...,e_n,e)$ is a path (or possibly a cycle) such that $P(p',t)=2$. This is forbidden by well-formedness. Hence, every generator can be visited at most once. As a consequence, the lexicographic order $(\#g,\#tk)$ (where $\#g$ is the number of non-visited generators in the diagram, and $\#tk$ the number of tokens in the diagram) strictly reduces with each rewrite. This finishes the proof of termination. ◀

Proof of Proposition 23. We are going to reason on every possible pairs of rewrite rules that can be applied from a single token state s. Notice first, that if the two rules are applied on two different terms of s, such that the rewriting of a term creates a copy of the other, they obviously commute, so $\begin{matrix} s & \rightsquigarrow & s_2 \\ \wr & & \wr \\ s_1 & \rightsquigarrow & s' \end{matrix}$.

In the case where $s = \alpha t + \beta t_1 + s_0$ such that $t_1 \rightsquigarrow s'$ and $t \rightsquigarrow \sum_i \lambda_i t_i$, we have:

$$\begin{tikzcd}
 & \alpha t + \beta s' + s_0 \arrow[r, rightsquigarrow] & \sum_i \alpha\lambda_i t_i + \beta s' + s_0 \arrow[d, rightsquigarrow] \\
s \arrow[ur, rightsquigarrow] \arrow[dr, rightsquigarrow] & & \\
 & (\alpha\lambda_1 + \beta)t_1 + \sum_{i\neq 1} \alpha\lambda_i t_i + s_0 \arrow[r, rightsquigarrow] & (\alpha\lambda_1 + \beta)s' + \sum_{i\neq 1} \alpha\lambda_i t_i + s_0
\end{tikzcd}$$

Then, we can, in the following, focus on pairs of rules applied on the same term. The term we focus on is obviously collision-free, by hypothesis and by preservation of collision-freeness by $\rightsquigarrow$.

Suppose the two rewrites are applied on tokens at positions e and e'. We may reason using the distance between the two edges.

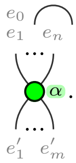
- the case $d(e, e') = 0$ would imply a collision, which is impossible by collision-freeness
- if $d(e, e') \geq 3$, the two rules still don't interfere, they commute (up to collisions which do not change the result)
- if $d(e, e') = 2$, there will be common collisions (i.e. collisions between tokens created by each of the diffusions), however, the order of application of the rules will not change the bits in the tokens we will apply a collision on, so the result holds
- if $d(e, e') = 1$, then the two tokens have to point to the same generator. If they didn't, (e, e') would form a path such that $|P((e, e'), t)| = 2$ which is forbidden by well-formedness.

We can then show the property for all generators:

Case e_0  e_1.

$$\begin{tikzcd}
(e_0 \downarrow x)(e_1 \downarrow x') \arrow[r, rightsquigarrow, "d"] \arrow[d, rightsquigarrow, "d"] & (e_1 \uparrow x)(e_1 \downarrow x') \arrow[d, rightsquigarrow, "c"] \\
(e_0 \downarrow x)(e_0 \uparrow x') \arrow[r, rightsquigarrow, "c"] & \langle x \mid x' \rangle
\end{tikzcd}$$

Case e_0  e_1: similar.

Case .

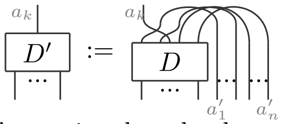
$$\begin{tikzcd}
e^{i\alpha x} \prod_{i\neq 1}(e_i \uparrow x) \prod_i (e'_i \downarrow x)(e'_1 \uparrow x') \arrow[dr, rightsquigarrow, "c"] & \\
(e_1 \downarrow x)(e'_1 \uparrow x') \arrow[u, rightsquigarrow, "d"] \arrow[d, rightsquigarrow, "d"] & \langle x \mid x' \rangle e^{i\alpha x} \prod_{i\neq 1}(e_i \uparrow x) \prod_{i\neq 1}(e'_i \downarrow x) \arrow[d, equal] \\
e^{i\alpha x'} \prod_i (e_i \uparrow x') \prod_{i\neq 1}(e'_i \downarrow x)(e_1 \downarrow x) \arrow[r, rightsquigarrow, "c"] & \langle x \mid x' \rangle e^{i\alpha x'} \prod_{i\neq 1}(e_i \uparrow x') \prod_{i\neq 1}(e'_i \downarrow x')
\end{tikzcd}$$

Case .

$$\begin{tikzcd}
\frac{1}{\sqrt{2}}\left((-1)^x (e_1 \downarrow x)(e_1 \uparrow x') + (e_1 \downarrow \neg x)(e_1 \uparrow x')\right) \arrow[dr, rightsquigarrow, "c"', "2"] & \\
(e_0 \downarrow x)(e_1 \uparrow x') \arrow[u, rightsquigarrow, "d"] \arrow[d, rightsquigarrow, "d"] & \frac{1}{\sqrt{2}}\left((-1)^x \langle x \mid x' \rangle + \langle \neg x \mid x' \rangle\right) \arrow[d, equal] \\
\frac{1}{\sqrt{2}}\left((-1)^{x'} (e_0 \downarrow x)(e_0 \uparrow x') + (e_0 \downarrow x)(e_0 \uparrow \neg x')\right) \arrow[r, rightsquigarrow, "c"', "2"] & \frac{1}{\sqrt{2}}\left((-1)^{x'} \langle x \mid x' \rangle + \langle x \mid \neg x' \rangle\right)
\end{tikzcd}$$

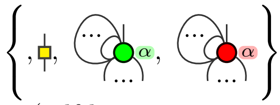
◀

Proof of Proposition 26. Let us first notice that, using the map/state duality, we have $(a_k \downarrow x) \rightsquigarrow^* \sum_{q=1}^{2^{m+n-1}} \lambda_q \prod_i (b_i \downarrow y_{i,q}) \prod_{i\neq k} (a_i \uparrow x_{i,q})$ in D iff we have $(a_k \downarrow x) \rightsquigarrow^* \sum_{q=1}^{2^{m+n-1}} \lambda_q \prod_i (b_i \downarrow$

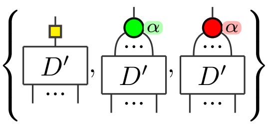
$y_{i,q})\prod_{i\neq k}(a'_i \downarrow x_{i,q})$ in D' where  . Hence, we can, w.l.o.g. consider in the following that $n=1$. We also notice that thanks to the confluence of the rewrite system, we can consider diagrams up to "topological deformations", and hence ignore cups and caps.

We then proceed by induction on the number N of “non-wire generators” (i.e. Z-spider, X-spiders and H-gates) of D, using the fact that the diagram is connected:

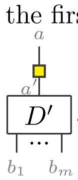
If $N=0$, then $D = |$, where the result is obvious.

If $N=1$, then $D \in$ . The result in this base case is then a straightforward verification (self-loops in green and red nodes simply give rise to collisions that are handled as expected).

For $N+1$, there exists D' with N non-wire generators and such that

$D \in$ 

(we should actually take into account the self loops, but they do not change the result). Let us look at the first two cases, since the last one can be induced by composition.

If $D =$ , then D' is necessarily connected, by connectivity of D. Then:

$$
\begin{aligned}
(a\downarrow x) &\rightsquigarrow \frac{(-1)^x}{\sqrt{2}}(a'\downarrow x)+\frac{1}{\sqrt{2}}(a'\downarrow \neg x)\\
&\rightsquigarrow^* \frac{(-1)^x}{\sqrt{2}}\sum_{q=1}^{2^m}\lambda_q\prod_{i=1}^{m}(b_i\downarrow y_{i,q})+\frac{1}{\sqrt{2}}\sum_{q=1}^{2^m}\lambda'_q\prod_{i=1}^{m}(b_i\downarrow y_{i,q})\\
&=\sum_{q=1}^{2^m}\frac{\lambda'_q+(-1)^x\lambda_q}{\sqrt{2}}\prod_{i=1}^{m}(b_i\downarrow y_{i,q})
\end{aligned}
$$

where by induction hypothesis

$$\llbracket D'\rrbracket\,|x\rangle = \sum_{q=1}^{2^m}\lambda_q\,|y_{1,q},...,y_{m,q}\rangle$$

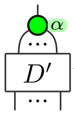
and

$$\llbracket D'\rrbracket\,|\neg x\rangle = \sum_{q=1}^{2^m}\lambda'_q\,|y_{1,q},...,y_{m,q}\rangle$$

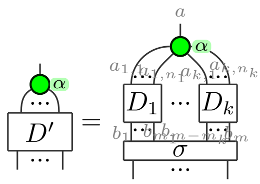
so:

$$
\begin{aligned}
\llbracket D\rrbracket\,|x\rangle = \llbracket D'\circ H\rrbracket\,|x\rangle = \llbracket D'\rrbracket\circ\llbracket H\rrbracket\,|x\rangle &= \llbracket D'\rrbracket\circ\left(\frac{(-1)^x}{\sqrt{2}}\,|x\rangle+\frac{1}{\sqrt{2}}\,|\neg x\rangle\right)\\
&=\frac{(-1)^x}{\sqrt{2}}\,\llbracket D'\rrbracket\,|x\rangle+\frac{1}{\sqrt{2}}\,\llbracket D'\rrbracket\,|\neg x\rangle=\sum_{q=1}^{2^m}\frac{\lambda'_q+(-1)^x\lambda_q}{\sqrt{2}}\,|y_{1,q},...,y_{m,q}\rangle
\end{aligned}
$$

which is the expected result.

Now, if $D =$ 

, we can decompose D' in its connected components:

$D =$ 

with D_i connected. Then:

$$
\begin{aligned}
(a \downarrow x) &\rightsquigarrow e^{i\alpha x} \prod_{\ell} \prod_{i} (a_{\ell,i} \downarrow x) \\
&\rightsquigarrow^* e^{i\alpha x} \prod_{\ell} \left(\sum_{q=1}^{2^{m_\ell+n_\ell-1}} \lambda_{q,\ell} \prod_{i \neq 1} (a_{\ell,i} \downarrow x)(a_{\ell,i} \uparrow x_{\ell,i,q}) \prod_{i} (b_{\ell,i} \downarrow y_{\ell,i,q}) \right) \\
&\rightsquigarrow^* e^{i\alpha x} \prod_{\ell} \left(\sum_{q=1}^{2^{m_\ell+n_\ell-1}} \lambda_{q,\ell} \delta_{x,x_{\ell,i,q}} \prod_{i} (b_{\ell,i} \downarrow y_{\ell,i,q}) \right) \\
&= e^{i\alpha x} \prod_{\ell} \left(\sum_{q=1}^{2^{m_\ell}} \lambda'_{q,\ell} \prod_{i} (b_{\ell,i} \downarrow y_{\ell,i,q}) \right) \\
&= e^{i\alpha x} \sum_{q_1=1}^{2^{m_1}} \dots \sum_{q_k=1}^{2^{m_k}} \lambda'_{q_1,1} \dots \lambda'_{q_k,k} \prod_{i} (b_{1,i} \downarrow y_{1,i,q_1}) \dots \prod_{i} (b_{k,i} \downarrow y_{k,i,q_k}) \\
&= \sum_{q=1}^{2^m} \lambda'_q \prod_{i} (b_i \downarrow y_{i,q})
\end{aligned}
$$

where the first is the diffusion through a Z-spider, and the second set of rewrites is the induction hypothesis applied to each connected component.

$$
\begin{aligned}
[\![D]\!] \, |x\rangle &= [\![(D_1 \otimes \dots \otimes D_k) \circ Z_k^1(\alpha)]\!] \, |x\rangle = ([\![D_1]\!] \otimes \dots \otimes [\![D_k]\!]) \circ [\![Z_k^1(\alpha)]\!] \, |x\rangle \\
&= e^{i\alpha x} ([\![D_1]\!] \otimes \dots \otimes [\![D_k]\!]) \circ |x, \dots, x\rangle = e^{i\alpha x} \, [\![D_1]\!] \, |x, \dots, x\rangle \otimes \dots \otimes [\![D_k]\!] \, |x, \dots, x\rangle \\
&= e^{i\alpha x} \left(\sum_{q_1}^{2^{m_1+n_1-1}} \lambda_{q_1,1} \, |y_{1,1,q_1}, \dots, y_{1,m_1,q_1}\rangle \left\langle x_{1,2,q_1}, \dots, x_{1,n_1,q_1} \,\middle|\, x, \dots, x \right\rangle \right) \otimes \\
&\qquad \dots \otimes \left(\sum_{q_k}^{2^{m_k+n_k-1}} \lambda_{q_k,k} \, |y_{k,1,q_1}, \dots, y_{k,m_1,q_k}\rangle \left\langle x_{k,2,q_k}, \dots, x_{k,n_1,q_k} \,\middle|\, x, \dots, x \right\rangle \right) \\
&= e^{i\alpha x} \left(\sum_{q_1}^{2^{m_1+n_1-1}} \lambda_{q_1,1} \prod_{i} \delta_{x,x_{1,i,q_1}} \, |y_{1,1,q_1}, \dots, y_{1,m_1,q_1}\rangle \right) \otimes \\
&\qquad \dots \otimes \left(\sum_{q_k}^{2^{m_k+n_k-1}} \lambda_{q_k,k} \prod_{i} \delta_{x,x_{k,i,q_k}} \, |y_{k,1,q_1}, \dots, y_{k,m_1,q_k}\rangle \right)
\end{aligned}
$$

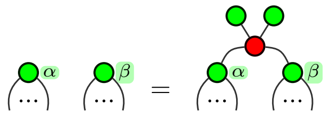
$$= e^{i\alpha x}\left(\sum_{q_1}^{2^{m_1}} \lambda'_{q_1,1} |y_{1,1,q_1}, ..., y_{1,m_1,q_1}\rangle\right) \otimes ... \otimes \left(\sum_{q_k}^{2^{m_k}} \lambda'_{q_k,k} |y_{k,1,q_1}, ..., y_{k,m_1,q_k}\rangle\right)$$

$$= \sum_{q=1}^{2^m} \lambda'_q |y_{1,q}, ..., y_{m,q}\rangle$$

where the third line is obtained by induction hypothesis, and all λ' match the ones obtained from the rewrite of token states. ◀

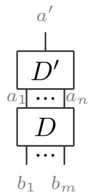
Proof of Lemma 27. There exist several methods to build a diagram D_f such that $[\![D_f]\!] = f$, using the universality of quantum circuits together with the map/state duality [7], or using normal forms [31]. The novelty here is that the diagram should be connected. This problem can be fairly simply dealt with:

Suppose we have such a D_f that has several connected components. We can turn it into an equivalent diagram that is connected. Let us consider two disconnected components of D_f. Each of these disconnected components either has at least one wire, or is one of {green node α, red node α}. In either case, we can use the rules of ZX ((I$_g$) or (H)) to force the existence of a green node. These green nodes in each of the connected components can be "joined" together like this:



It is hence possible to connect every different connected components of a diagram in a way that preserves the semantics. ◀

Proof of Proposition 28. Using Lemma 27, there exists a connected ZX-diagram D' with $\mathcal{I}(D') = [a']$ and such that $[\![D']\!]\,|0\rangle = \sum_{q=1}^{2^n} \lambda_q |x_{1,q}, ..., x_{n,q}\rangle$. Consider now a derivation from the token state $(a' \downarrow 0)$ in $D \circ D'$:

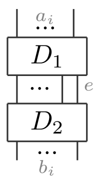


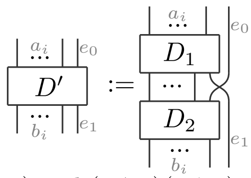
$(a' \downarrow 0) \rightsquigarrow^* \sum_{q=1}^{2^n} \lambda_q \prod_{i=1}^n (a_i \downarrow x_{i,q})$
and
$(a' \downarrow 0) \rightsquigarrow^* \sum_{q=1}^{2^m} \lambda'_q \prod_{i=1}^m (b_i \downarrow y_{i,q})$

The first run comes from Proposition 26 on D' which is connected. The second run results from Proposition 26 on $D \circ D'$ which is also connected. The proposition also gives us that:

$$[\![D]\!] \circ \left(\sum_{q=1}^{2^n} \lambda_q |x_{1,q}, ..., x_{n,q}\rangle\right) = [\![D]\!] \circ [\![D']\!] \circ |0\rangle = [\![D \circ D']\!] \circ |0\rangle = \sum_{q=1}^{2^m} \lambda'_q |y_{1,q}, ..., y_{m,q}\rangle$$

Finally, by confluence in $D \circ D'$, we get $\sum_{q=1}^{2^n} \lambda_q \prod_{i=1}^n (a_i \downarrow x_{i,q}) \rightsquigarrow^* \sum_{q=1}^{2^m} \lambda'_q \prod_{i=1}^m (b_i \downarrow y_{i,q})$ in D. ◀

Proof of Theorem 29. First, let us single out e in the diagram $D =$ 

. We can build a second diagram by cutting e in half and seeing each piece of wire as an input and an output: 

. We can easily see that a rewriting of the token states $(e \downarrow 0)(e \uparrow 0)$ and $(e \downarrow 1)(e \uparrow 1)$ in D correspond step by step to a rewriting of the token states $(e_0 \downarrow 0)(e_1 \uparrow 0)$ and $(e_0 \downarrow 1)(e_1 \uparrow 1)$ in D'. We can then focus on D', whose interpretation is taken to be

$$[\![D']\!] = \sum_{q=1}^{2^{m+n+2}} \lambda'_q \left| y'_{1,q}, ..., y'_{m+1,q} \middle\rangle\!\middle\langle x'_{1,q}, ..., x'_{n+1,q} \right|$$

such that

$$(id^{\otimes m} \otimes \langle 0|) \circ [\![D']\!] \circ (id^{\otimes n} \otimes |0\rangle) + (id^{\otimes m} \otimes \langle 1|) \circ [\![D']\!] \circ (id^{\otimes n} \otimes |1\rangle) = [\![D]\!]$$

from which we get:

$$\begin{aligned} [\![D]\!] = &\sum_{q=1}^{2^{m+n+2}} \lambda'_q \delta_{0,y'_{m+1,q}} \delta_{0,x'_{n+1,q}} \left| y'_{1,q}, ..., y'_{m,q} \middle\rangle\!\middle\langle x'_{1,q}, ..., x'_{n,q} \right| \\ &+ \sum_{q=1}^{2^{m+n+2}} \lambda'_q \delta_{1,y'_{m+1,q}} \delta_{1,x'_{n+1,q}} \left| y'_{1,q}, ..., y'_{m,q} \middle\rangle\!\middle\langle x'_{1,q}, ..., x'_{n,q} \right| \end{aligned}$$

We now have to consider two cases:

- D' is still connected: By Proposition 26, for $x \in \{0,1\}$:

$$\begin{aligned} (e_0 \downarrow x)(e_1 \uparrow x) &\rightsquigarrow^* \sum_{q=1}^{2^{m+n+2}} \lambda'_q \delta_{x,x'_{n+1,q}} \prod_i (a_i \uparrow x'_{i,q}) \prod_i (b_i \downarrow y'_{i,q})(e_1 \downarrow y'_{m+1,q})(e_1 \uparrow x) \\ &\rightsquigarrow \sum_{q=1}^{2^{m+n+2}} \lambda'_q \delta_{x,y'_{m+1,q}} \delta_{x,x'_{n+1,q}} \prod_i (a_i \uparrow x'_{i,q}) \prod_i (b_i \downarrow y'_{i,q}) \end{aligned}$$

  We hence have

$$\begin{aligned} (e_0 \downarrow 0)(e_1 \uparrow 0) \rightsquigarrow^* t_0 = \sum_{q=1}^{2^{m+n+2}} \lambda'_q \delta_{0,y'_{m+1,q}} \delta_{0,x'_{n+1,q}} \prod_i (a_i \uparrow x'_{i,q}) \prod_i (b_i \downarrow y'_{i,q}) \\ (e_0 \downarrow 1)(e_1 \uparrow 1) \rightsquigarrow^* t_1 = \sum_{q=1}^{2^{m+n+2}} \lambda'_q \delta_{1,y'_{m+1,q}} \delta_{1,x'_{n+1,q}} \prod_i (a_i \uparrow x'_{i,q}) \prod_i (b_i \downarrow y'_{i,q}) \end{aligned}$$

  so $t_0 + t_1$ corresponds to the interpretation of D.
- D' is now disconnected: Since D was connected, the two connected components of D were connected through e. Hence, D' only has two connected components, one connected to e_0 and the other to e_1. By applying Proposition 26 to both connected components, we get the desired result.

◀

B Proof of Section 4

Proof of Proposition 35. By a straightforward induction on $\rightsquigarrow_{\mathbf{sop}}$. ◀

Proof of Proposition 36. Let $D \in \mathbf{ZX}$ and $s \in \mathbf{tkS_{SOP}}(D)$ such that its $B_j \in \{0, 1, y\}_{y \in V}$ for all j. Note that all collisions at worst do not change the size of the term (at best reduce the size). Indeed, we turn two tokens into at most two terms in the phase polynomial, since $\frac{z}{2}(\widehat{B_{j_1} \oplus B_{j_2}} = \frac{z}{2}(B_{j_1} + B_{j_2} - 2B_{j_1}B_{j_2}) = \frac{z}{2}(B_{j_1} + B_{j_2})$ because we work modulo 1 in the phase polynomial.

Hence, since a rewrite step consists in a diffusion step followed by some collision rule, showing the result only for diffusions is enough.

- Diffusions through Cups and Caps do not change the size.
- A diffusion through H adds a single term in the phase polynomial. However, since H is in the diagram, $\Delta(D) \geq 2$, so the proposition holds.
- A diffusion through a Green-spider with arity δ adds $\delta - 2$ tokens, and a single term in the phase polynomial. However, $\delta \leq \Delta(D)$.

◀

C Proof of Section 5

Proof of Theorem 39. Diffusion rules are trivial. Beware in the case of the Ground, as the CPM will produce a cup, the $\rightsquigarrow_{\fourier}$ does not produce a new token when applying the Trace-Out rule, meanwhile the $\rightsquigarrow_{\mathbf{sop}}$ machine will do two rewriting rules to pass through the cup. ◀