Geometry of Interaction for ZX-Diagrams

Kostia Chardonnet
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France
Université de Paris, CNRS, IRIF, F-75006, Paris, France

Benoît Valiron
Université Paris-Saclay, CNRS, CentraleSupélec, ENS Paris-Saclay, LMF,
91190, Gif-sur-Yvette, France

Renaud Vilmart
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, LMF, 91190, Gif-sur-Yvette, 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.MFCS.2021.30

Related Version *Full Version*: `https://hal.archives-ouvertes.fr/hal-03154573/`

Funding This work was supported in part by the French National Research Agency (ANR) under the research project SoftQPRO ANR-17-CE25-0009-02, and by the DGE of the French Ministry of Industry under the research project PIA-GDN/QuantEx P163746-484124.

1 Introduction

Quantum computing is a model of computation where data is stored on the state of particles governed by the laws 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 [29, 31, 17].

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*: 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* [30] for describing quantum operations with quantum gates, the quantum version of Boolean gates. Although the pervasive model for quantum computation, quantum circuits' operational semantics is only given in an intuitive manner. A quantum circuit is understood as some sequential, low-level assembly language where quantum gates are opaque black-boxes. In particular, quantum circuits do not natively feature any formal operational semantics giving rise to abstract reasoning, equational theory or well-founded rewrite system.

From a denotational perspective, quantum circuits are literal description of tensors and applications of linear operators. These can be described with the historical matrix interpretation [30], or with the more recent sum-over-paths semantics [1, 6] – this can be

46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021).
Editors: Filippo Bonchi and Simon J. Puglisi; Article No. 30; pp. 30:1–30:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

regarded as a *wave-style semantics*. In such a semantics, the state of all of the quantum bits of the memory is mathematically represented as a vector in a (finite dimensional) Hilbert space: 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.

In recent years, an alternative model of quantum computation with better formal properties than quantum circuits has 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*. Unlike the quantum circuit framework, the ZX-Calculus comes with a sound and complete [24, 33], well-defined equational theory on a small set of canonical generators making it possible to reason on quantum computation by means of local graph rewriting.

The canonical semantics of a ZX diagram consists in a linear operator. This operator can be represented as a matrix or through the more recent sum-over-path semantics [35]. But in both cases, these semantics give a purely functional, *wave-style* interpretation to the diagram. Nonetheless, this graphical language – and its equational theory – has been shown to be amenable to many extensions and is being used in a wide spectrum of applications ranging from quantum circuit optimization [14, 4], verification [25, 15, 13] and representation such as MBQC patterns [16] or error-correction [12, 11].

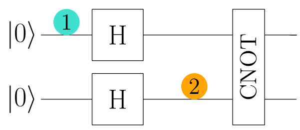
The standard models for both quantum circuits and ZX-Calculus is therefore based on a wave-style interpretation. An alternative operational interpretation of quantum circuit following a *particle-style* semantics has recently been investigated in the literature [9]. In this model, quantum bits are intuitively seen as *tokens* flowing inside the wires of the circuit. Formally, a quantum circuit is interpreted as a token-based automata, based on Geometry of Interaction (GoI) [21, 20, 19, 22]. Among its many instantiations, GoI can be seen as a procedure to interpret a proof-net [23] – graphical representation of proofs of linear logic [18] – as a token-based automaton [10, 2]. The flow of a token inside a proof-net characterises an invariant of the proof – its computational content. This framework is used in [9] 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.

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 particle-style perspective.

In this paper, we aim at giving a novel insight on the computational content of a ZX term in an asynchronous way, emphasizing the graph-like behavior of a ZX-diagram.

Following the idea of using a token machine to exhibit the computational content of a proof-net or a quantum circuit, we present in this paper a token machine for the ZX-Calculus. To exemplify the versatility of the approach, we show how to extend it to mixed processes [8, 5]. To assess the validity of the semantics, we show how it links to the standard interpretation of ZX-diagrams. While the standard interpretation of ZX-diagrams proceeds with diagram decomposition as tensors and products of matrices, the tokens flowing inside the diagram really exploits the connectivity of the diagram.

This ability illustrates one fundamental difference between our approach and the one in [9]. The latter follows a *classical control* approach: if qubits can be in superposition, each qubit inhabits a token sitting in *one single* position in the circuit. For instance, on the circuit on the right, the state of the two tokens



$|\bullet\bullet\rangle$ is $\frac{\sqrt{2}}{2}(|00\rangle + |10\rangle)$. Although the two tokens can be regarded as being in superposition, their *position* is not. In our system, tokens and positions can be superposed. The second fundamental difference lies in the *asynchronicity* of our token-machine. Unlike [9], we rely on the canonical generators of ZX-diagrams: tokens can travel through these nodes in an asynchronous manner. For instance, in the above circuit the orange token has to wait for the blue token before crossing the CNOT gate. As illustrated in Table 1, in our system one token can interact with multi-wire nodes. Finally, as formalized in Theorem 27, a third difference is that compared to [9], the token-machine we present is *non-oriented*: in the circuit above, tokens have to start on the left and flow towards the right of the circuit whereas our system is agnostic on where tokens initially "start".

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.

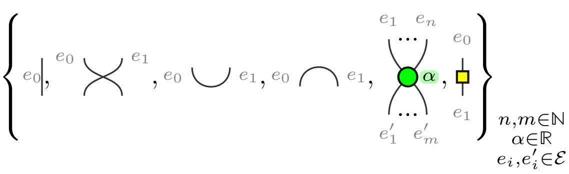
In Section 3 we present the actual asynchronous token machine and its semantics and show that it is sound and complete with regard to the standard interpretation of ZX-diagrams. Finally, in Section 4 we present an extension of the ZX-Calculus to mixed processes and adapt the token machine to take this extension into account. 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.

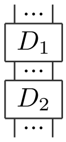
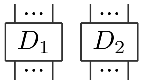
2.1 Pure Operators

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:



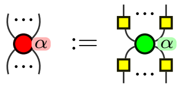
We shall be using the following labeling convention: wires (edges) are labeled with e_i, taken from an infinite set of labels $\mathcal{E}$. We take for granted that distinct wires have distinct labels. The real number α attached to the green spiders is called the *angle*. ZX-diagrams are read top-to-bottom: 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.

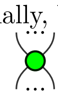
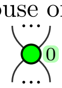
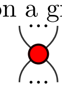
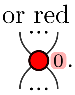
ZX-primitives can be composed either sequentially or in parallel:

$D_2 \circ D_1 :=$  $D_1 \otimes D_2 :=$ 

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.

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



▶ **Convention 2.** We write σ for a permutation of wires, i.e any diagram generated by $\left\{ \quad \right\}$ 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:  :=  and  := .

2.2 Standard Interpretation

We understand ZX-diagrams as linear operators through the *standard interpretation*. Informally, wires are interpreted with the two-dimensional Hilbert space, with orthonormal basis written as $\{|0\rangle, |1\rangle\}$, in Dirac notation [30]. Vectors of the form $|.\rangle$ (called "*kets*") are considered as columns vector, 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. [30].

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

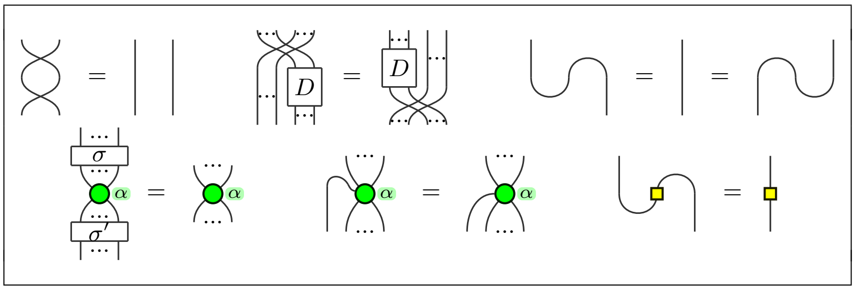
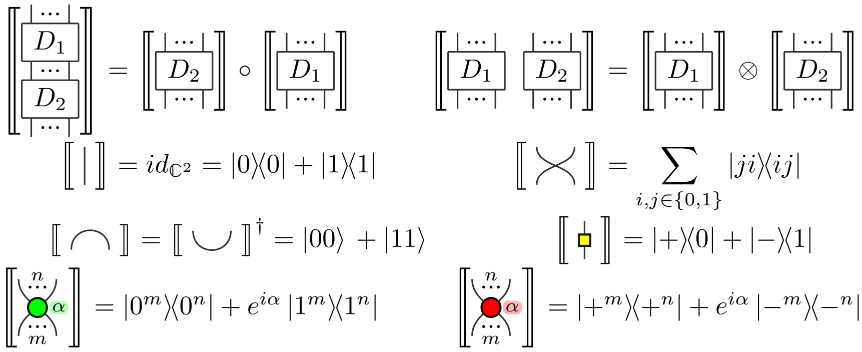


Figure 1 Connectivity rules. D represents any ZX-diagram, and σ, σ' any permutation of wires.

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.



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. [34] 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 **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, 26, 24, 27, 28, 33].

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.

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 to be *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 exists axiomatizations 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** (Choi-Jamiołkowski). *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* $k \leq n$*).*

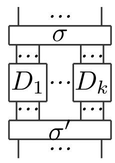
To see how this can be true, simply add cups or caps to turn input edges to output edges (or vice versa), and use the fact that we work modulo the rules of Figure 1.

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 decomposition. 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.*

We can also consider the notions of paths, distance and cycles of usual multi-graphs. We denote $\mathrm{Paths}(e_0, e_n)$ the set of paths from edge e_0 to e_n. We denote $\mathrm{Paths}(D)$ (resp. $\mathrm{Cycles}(D)$) the set of paths (resp. cycles) of diagram D. For a path p, we denote $|p|$ its length. We denote $d(e_0, e_n)$ the distance i.e. the length of the shortest path between e_0 and e_n.

3 A Token Machine for ZX-diagrams

Inspired by the Geometry of Interaction [21, 20, 19, 22] and the associated notion of token machine [10, 2] for proof nets [23], 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 5** (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 6.** In token states, the sum $(+)$ stands for the superposition while the product stands 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 $(t_1 + t_2)t_3 = (t_1 t_2) + (t_1 t_3)$, 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 $t_1 t_2$ is the same as $t_2 t_1$. Notice that 0 is an absorbing element for the product $(0 \times t = 0)$ and that 1 is a neutral element for the same operation $(1 \times t = t)$.

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 5.

▶ **Definition 7** (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 6,

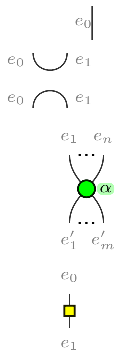
$$\sum_i \alpha_i t_{i,1} \cdots t_{i,j} \cdots t_{i,n_i} \rightsquigarrow_X \sum_i \alpha_i t_{i,1} \cdots \left(\sum_k \beta_k t'_k\right) \cdots t_{i,n_i}$$

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.

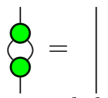
Finally, as it is customary for rewrite systems, if $(\to)$ is a step in a transition system, $(\to^*)$ stands for the reflexive, transitive closure of $(\to)$.

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.

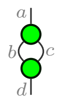
Table 1 Asynchronous token-state evolution, for all $x, b \in \{0,1\}$.



$(e_0 \downarrow x)(e_0 \uparrow x) \rightsquigarrow_c 1 \quad (e_0 \downarrow x)(e_0 \uparrow \neg x) \rightsquigarrow_c 0$	(Positive/Negative Collision)
$(e_b \downarrow x) \rightsquigarrow_d (e_{\neg b} \uparrow x)$	($\smile$-diffusion)
$(e_b \uparrow x) \rightsquigarrow_d (e_{\neg b} \downarrow x)$	($\frown$ -diffusion)
$(e_k \downarrow x) \rightsquigarrow_d e^{i\alpha x} \prod_{i \neq k} (e_i \uparrow x) \prod_j (e'_j \downarrow x)$ $(e'_k \uparrow x) \rightsquigarrow_d e^{i\alpha x} \prod_{j \neq k} (e'_j \downarrow x) \prod_i (e_i \uparrow x)$	(Green spider -Diffusion)
$(e_0 \downarrow x) \rightsquigarrow_d (-1)^x \frac{1}{\sqrt{2}} (e_1 \downarrow x) + \frac{1}{\sqrt{2}} (e_1 \downarrow \neg x)$ $(e_1 \uparrow x) \rightsquigarrow_d (-1)^x \frac{1}{\sqrt{2}} (e_0 \uparrow x) + \frac{1}{\sqrt{2}} (e_0 \uparrow \neg x)$	(Hadamard -Diffusion)

▶ **Example 8.** Consider the equality given by the ZX equational theories: .

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 rewritten at each step. This is what we get when giving the priority to collisions:



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

Notice that the collision $(c \uparrow 0)(c \downarrow 0)$ rewrites to 1, and therefore the product $(d \downarrow 0) \times 1 = (d \downarrow 0)$. If however we decide to ignore the priority of collisions, we may end up with a non-terminating run, unable to converge to $(d \downarrow 0)$:

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

We therefore set a rewriting strategy as follows.

▶ **Definition 9** (Collision-Free). *A token state s of* $\mathbf{tkS}(D)$ *is called* collision-free *if for all $s' \in \mathbf{tkS}(D)$, we have $s \not\rightsquigarrow_c s'$.*

▶ **Definition 10** (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, $t \rightsquigarrow u$ if and only if there exists t' such that $t \rightsquigarrow_d t' \rightsquigarrow_c^* u$ and u is collision-free.*

In [9], a token arriving at an input of a gate is blocked until all the inputs of the gates are populated by a token, at which point all the tokens go through at once (while obviously changing the state). The control is purely classical: it is causal. In our approach, the state of the system is global and there is no explicit notion of qubit. Instead, tokens collect the operation that is to be applied to the input qubits.

3.2 Strong Normalization and Confluence

The token machine Rewrite System of Definition 10 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 11** (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:*

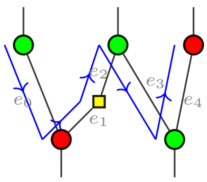
$$P(p,t) = \begin{cases} 1 & \text{if } e \in p \text{ and } e \text{ is } d\text{-oriented} \\ -1 & \text{if } e \in p \text{ and } e \text{ is } \neg d\text{-oriented} \\ 0 & \text{if } e \notin p \end{cases}$$

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

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

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

▶ **Example 12.** 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 13** (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 14** (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 15, this property is in fact stronger.

▶ **Proposition 15** (Full Characterisation of Well-Formed Terms). *Let D be a ZX-diagram, and $s \in \mathbf{tkS}(D)$ 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 16** (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 17** (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 18** (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 19** (Termination of well-formed, cycle-balanced token state)**.** *Let D be a ZX-diagram, and $s \in \mathbf{tkS}(D)$ 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 20** (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 21** (Confluence)**.** *Let D be a ZX-diagram. The rewrite system $\rightsquigarrow$ is confluent for well-formed, collision-free and cycle-balanced token states.*

▶ **Corollary 22** (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 23** (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: $(a_k \downarrow x) \rightsquigarrow^* \displaystyle\sum_{q=1}^{2^{m+n-1}} \lambda_q \prod_i (b_i \downarrow y_{i,q}) \prod_{i \neq k} (a_i \uparrow x_{i,q})$

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 $|x\rangle$.

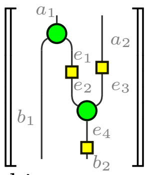
Proposition 23 can be made more general. However, we first need the following result on ZX-diagrams:

▶ **Lemma 24** (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 25** (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: $$\sum_{q=1}^{2^n} \lambda_q \prod_{i=1}^{n} (a_i \downarrow x_{i,q}) \rightsquigarrow^* \sum_{q=1}^{2^m} \lambda'_q \prod_{i=1}^{m} (b_i \downarrow y_{i,q})$$

▶ **Example 26** (CNOT). In the ZX-Calculus, the CNOT-gate (up to some scalar) can be constructed as follows:  $= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

On classical inputs, this gate applies the NOT-gate on the second bit if and only if the first bit is at 1. Therefore if we apply the state $|10\rangle$ to it we get $\frac{1}{\sqrt{2}}|11\rangle$.

We demonstrate how the token machine can be used to get this result. Following Proposition 25, we start by initialising the Token Machine in the token state $(a_1 \downarrow 1)(a_2 \downarrow 0)$, matching the input state $|10\rangle$.

We underline each step that is being rewritten, and take the liberty to sometimes do several rewrites in parallel at the same time.

$$\begin{aligned}
\underline{(a_1 \downarrow 1)}(a_2 \downarrow 0) &\rightsquigarrow_d (b_1 \downarrow 1)(e_1 \downarrow 1)\underline{(a_2 \downarrow 0)} \rightsquigarrow_d (b_1 \downarrow 1)(e_1 \downarrow 1)\tfrac{1}{\sqrt{2}}\Big(\underline{(e_3 \downarrow 0)} + \underline{(e_3 \downarrow 1)}\Big) \\
&\rightsquigarrow_d \tfrac{1}{\sqrt{2}}(b_1 \downarrow 1)\underline{(e_1 \downarrow 1)}\Big((e_2 \uparrow 0)(e_4 \downarrow 0) + (e_2 \uparrow 1)(e_4 \downarrow 1)\Big) \\
&\rightsquigarrow_d \tfrac{1}{2}(b_1 \downarrow 1)\Big(\underline{(e_2 \downarrow 0)} - \underline{(e_2 \downarrow 1)}\Big)\Big(\underline{(e_2 \uparrow 0)}(e_4 \downarrow 0) + (e_2 \uparrow 1)(e_4 \downarrow 1)\Big) \\
&\rightsquigarrow_c^2 \tfrac{1}{2}(b_1 \downarrow 1)\Big((e_4 \downarrow 0) + \big(\underline{(e_2 \downarrow 0)} - \underline{(e_2 \downarrow 1)}\big)\underline{(e_2 \uparrow 1)}(e_4 \downarrow 1)\Big) \\
&\rightsquigarrow_c^2 \tfrac{1}{2}(b_1 \downarrow 1)\Big(\underline{(e_4 \downarrow 0)} - \underline{(e_4 \downarrow 1)}\Big) \\
&\rightsquigarrow_d \tfrac{1}{2\sqrt{2}}(b_1 \downarrow 1)\Big((b_2 \downarrow 0) + (b_2 \downarrow 1) - (b_2 \downarrow 0) + (b_2 \downarrow 1)\Big) \\
&= \tfrac{1}{\sqrt{2}}(b_1 \downarrow 1)(b_2 \downarrow 1)
\end{aligned}$$

The final token state corresponds to $\frac{1}{\sqrt{2}}|11\rangle$, as described by Proposition 25. Notice that during the run, each invariants presented before holds and that due to confluence we could have rewritten the tokens in any order and still obtain the same result.

This proposition is a direct generalization of Proposition 23. It shows we can compute the output of a diagram provided a particular input state. We can also recover the semantics of the whole operator by initialising the starting token state in a particular configuration.

▶ **Theorem 27** (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 terminates by Corollary 22). Then:*

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

▶ **Example 28.** If we take back the diagram from Example 26 and decide to initialize any wire e of the diagram in the state $(e\downarrow 0)(e\uparrow 0)+(e\downarrow 1)(e\uparrow 1)$ and apply the rewriting as in Theorem 27 we in fact end up with the token state $\frac{1}{\sqrt{2}}\Big((a_1\uparrow 0)(a_2\uparrow 0)(b_1\downarrow 0)(b_2\downarrow 0)+(a_1\uparrow 0)(a_2\uparrow 1)(b_1\downarrow 0)(b_2\downarrow 1)+(a_1\uparrow 1)(a_2\uparrow 0)(b_1\downarrow 1)(b_2\downarrow 1)+(a_1\uparrow 1)(a_2\uparrow 1)(b_1\downarrow 1)(b_2\downarrow 0)\Big)$ which matches the actual matrix of the standard interpretation.

▶ Remark 29. At this point, it is legitimate to wonder about the benefits of the token machine over the standard interpretation for computing the semantics of a diagram. Let us first notice that when computing the semantics of a diagram à la Theorem 27, we get in the token state one term per non-null entry in the associated matrix (the one obtained by the standard interpretation).

We can already see that the token-based interpretation can be interesting if the matrix is sparse, the textbook case being Z_n^n whose standard interpretation requires a $2^n \times 2^n$ matrix, while the token-based interpretation only requires two terms (each with $2n$ tokens).

Secondly, we can notice that we can "mimic" the standard interpretation with the token machine. Consider a diagram decomposed as a product of slices (tensor product of generators) for the standard interpretation. Then, for the token machine, without going into technical details, we can follow the strategy that consists in moving token through the diagram one slice at a time. This essentially computes the matrix associated with each slice and its composition.

The point of the token machine however, is that it is versatile enough to allow for more original strategies, some of which may have a worst complexity, but also some of which may have a better one.

4 Extension to Mixed Processes

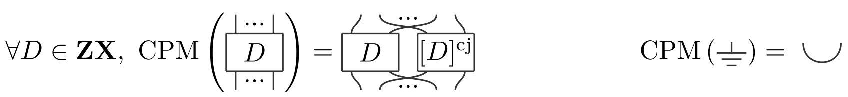
The token machine presented so far worked for so-called *pure* quantum processes i.e. with no interaction with the environment. To demonstrate how generic our approach is, we show how to adapt it to the natural extension of *mixed* processes, represented with completely positive maps (CPM). This in particular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

4.1 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], that intuitively enforces the state of the wire to be classical. We denote with $\mathbf{ZX}^{⏚}$ the set of diagrams obtained by adding ⏚ this generator.

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 [32]. We define the map (conveniently named) CPM as the map that preserves compositions $(\_\circ\_)$ and $(\_\otimes\_)$ and such that:

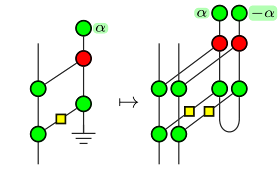


Where $[D]^{\mathrm{cj}}$ is D where every angle α has been changed to $-\alpha$.

With respect to what happens to edge labels, notice that every edge in D can be mapped to 2 edges in $\mathrm{CPM}(D)$. We propose that label e induces label e in the first copy, and $\overline{e}$ in the second, e.g, for the identity diagram: $|_{e_0} \longmapsto |_{e_0}\,|_{\overline{e_0}}$

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 30.** A $\mathbf{ZX}^{\text{⏚}}$-diagram and its associated CPM construction is shown on the right (without names on the wires for simplicity).



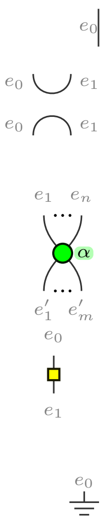
4.2 Token Machine for Mixed Processes

We now aim to adapt the token machine to $\mathbf{ZX}^{\text{⏚}}$, the formalism for completely positive maps. To do so we give an additional state to each token to mimic the evolution of two token on $\mathrm{CPM}(D)$.

▶ **Definition 31.** *Let D be a ZX-diagram. A ⏚-token is a quadruplet $(p,d,x,y) \in \mathcal{E}(D) \times \{\downarrow,\uparrow\} \times \{0,1\} \times \{0,1\}$. We denote the set of ⏚-tokens on D by $\mathbf{tk}^{\text{⏚}}(D)$. A ⏚-token-state is then a multivariate polynomial over $\mathbb{C}$, evaluated in $\mathbf{tk}^{\text{⏚}}(D)$. We denote the set of ⏚-token-states on D by $\mathbf{tkS}^{\text{⏚}}(D)$*

In other words, the difference with the previous machine is that tokens here have an additional state (e.g. y in $(a \downarrow x, y)$). The rewrite rules are given in appendix in Table 2.

Table 2 The rewrite rules for $\rightsquigarrow_{\text{⏚}}$, where δ is the Kronecker delta.



$$(e_0 \downarrow x,y)(e_0 \uparrow x',y') \rightsquigarrow_c \delta_{x,x'}\delta_{y,y'} \qquad \text{(Collision)}$$

$$(e_b \downarrow x,y) \rightsquigarrow_d (e_{\neg b} \uparrow x,y) \qquad (\cup\text{-diffusion})$$

$$(e_b \uparrow x,y) \rightsquigarrow_d (e_{\neg b} \downarrow x,y) \qquad (\cap\text{-diffusion})$$

$$(e_k \downarrow x,y) \rightsquigarrow_d e^{i\alpha(x-y)} \prod_{j\neq k}(e_j \uparrow x,y) \prod_{j}(e'_j \downarrow x,y)$$

$$(e'_k \uparrow x,y) \rightsquigarrow_d e^{i\alpha(x-y)} \prod_{j}(e_j \uparrow x,y) \prod_{j\neq k}(e'_j \downarrow x,y) \qquad (\text{green spider-Diffusion})$$

$$(e_0 \downarrow x,y) \rightsquigarrow_d \frac{1}{2} \sum_{z,z' \in\{0,1\}} (-1)^{xz+yz'}(e_1 \downarrow z,z')$$

$$(e_1 \uparrow x,y) \rightsquigarrow_d \frac{1}{2} \sum_{z,z' \in\{0,1\}} (-1)^{xz+yz'}(e_0 \uparrow z,z') \qquad (\text{yellow box-Diffusion})$$

$$(e_0 \downarrow x,y) \rightsquigarrow_d \delta_{x,y} \qquad \text{(Trace-Out)}$$

It is possible to link this formalism back to the pure token-states, using the existing CPM construction for ZX-diagrams. We extend this map by $\mathrm{CPM} : \mathbf{tkS}^{\text{⏚}}(D) \to \mathbf{tkS}(\mathrm{CPM}(D))$, defined as: $\sum_{q=1}^{2^{m+n}} \lambda_q \prod_j (p_j, d_j, x_{j,q}, y_{j,q}) \mapsto \sum_{q=1} \lambda_q \prod_j (p_j, d_j, x_{j,q})(\overline{p_j}, d_j, y_{j,q})$

Since $\mathrm{CPM}(D)$ can be seen as two copies of D where ⏚ is replaced by ◡ , 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 states x and y represent the states of the two corresponding tokens.

We can then show that this rewriting system is consistent:

▶ **Theorem 32.** *Let D be a $\mathbf{ZX}^{⏚}$-diagram, and $t_1, t_2 \in \mathbf{tkS}^{⏚}(D)$. Then whenever $t_1 \rightsquigarrow_{⏚} t_2$ we have $\mathrm{CPM}(t_1) \rightsquigarrow^{\{1,2\}} \mathrm{CPM}(t_2)$.*

The notions of polarity, well-formedness and cycle-balancedness can be adapted, and we get strong normalization (Theorem 19), confluence (Corollary 21), and uniqueness of normal forms (Corollary 22) for well-formed and cycle-balanced token states.

5 Conclusion and Future Work

In this paper, we presented a novel *particle-style* semantics for ZX-Calculus. Based on a token-machine automaton, it emphasizes the *asynchronicity* and *non-orientation* of the computational content of a ZX-diagram. Compared to existing token-based semantics of quantum computation such as [9], our proposal furthermore support decentralized tokens where the position of a token can be in superposition.

As 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 [9] where some form of synchronicity is enforced.

Our token machines give us a new way to look at how a ZX-diagram computes with a more local, operational approach. This could lead to extensions of the ZX-Calculus with more expressive logical and computational constructs, such as recursion.

As a final remark, we notice that this formalism naturally extends to other graphical languages for qubit quantum computation, and even for tensor networks. It suffices to adapt the diffusion rewriting steps to the generators at hand, which is always possible in the setting of finite dimensional Hilbert spaces, and if needs be to adapt the states in tokens to the dimension of the wire they go through (e.g. if a wire in a tensor network is of dimension 4, the state spans $\{0, 1, 2, 3\}$).

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. 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, November 2012. doi:10.2168/LMCS-8(4:14)2012.

9 Dal Lago, Ugo and Faggian, Claudia and Valiron, Benoît and Yoshimizu, Akira. The geometry of parallelism: Classical, probabilistic, and quantum effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, page 833–845, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3009837.3009859.

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

11 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, 2019. 12 pages + appendices. URL: https://hal.archives-ouvertes.fr/hal-02413388.

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

13 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.

14 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.

15 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.

16 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.

17 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.

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

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

20 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.

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

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

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

24 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. doi:10.1145/3209108.3209128.

25 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.

26 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. doi:10.1145/3209108.3209131.

27 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. doi:10.1145/3209108.3209139.

28 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.

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

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

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

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

33 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.

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

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