From the proceedings of Trends in Linear Logic and Applications 2021 (TLLA 2021, eds E. Pimentel, Ch. Retoré)
https://hal.archives-ouvertes.fr/TLLA2021

Geometry of Interaction for ZX-Diagrams

Kostia Chardonnet* Benoît Valiron† Renaud Vilmart‡

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 pure ZX-Calculus 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.

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 or quantum programming languages. 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* [NC02] 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 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. 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

*Univ. Paris Saclay - LMF, Univ. Paris - IRIF
†Univ. Paris Saclay - LMF, CentraleSupélec
‡Univ. Paris Saclay - LMF, Inria - DEDUCTEAM

"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 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 [Amy18, CBB+]. Still based on the original mathematical representation of state-as-a-vector, the sum-over-path of a quantum circuit model the action of the circuit using 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 circuits [LFVY16] is based on Geometry of Interaction (GoI) [Gir89]. Among its many instantiations, GoI can be seen as a procedure to intepret a proof-net [Gir96] —graphical representation of proofs of linear logic [Gir87]— as a token-based automaton [DR99]. The flow of a token inside a proof-net characterizes an invariant of the proof —its computational content. This framework is used in [LFVY16] 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 [LFVY16], 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 [CD11]. 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 quantum circuit framework, ZX-Calculus also comes with a small set of canonical generators with a well-defined semantics.

Reasoning 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 [DKPVDW20], verification [Hil11] or error-correction [dBH20].

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 semantics, similarly to what has been done for quantum circuits.

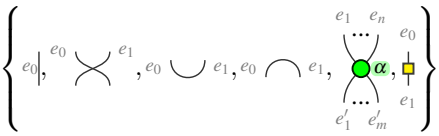
Following the idea of applying a token machine to proof-nets in order to study its computational content, we present in this paper a token machine for the ZX-Calculus. We show how it links to the standard interpretation of ZX-diagrams. While the standard interpretation of ZX-diagrams proceeds with conventional 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.

2 The ZX-Calculus

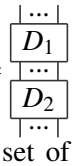
The ZX-Calculus is a powerful graphical language for reasoning about quantum computation introduced by Bob Coecke and Ross Duncan [CD11]. A term in this language is a graph —called a *string diagram*— built from a core set of primitives. In the standard interpretation of ZX-Calculus, a string diagram is interpreted as a matrix. The language is equipped with an equational theory preserving the standard interpretation.

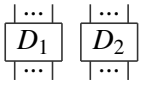
2.1 Pure Operators

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, with $n,m \in \mathbb{N}, \alpha \in \mathbb{R}, e_i, e_i' \in \mathscr{E}$:



We shall be using the following labeling convention: wires (edges) are labeled with e_i, taken from an infinite set of labels $\mathscr{E}$. 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. By abuse of notation a green node with no explicit parameter holds the angle 0. We write $\mathscr{E}(D)$ for the set of edge labels in the diagram D, and $\mathscr{I}(D)$ (resp. $\mathscr{O}(D)$) for the list of input edges (resp. output edges) of D. ZX-primitives can be composed either sequentially or in parallel :

$D_2 \circ D_1 :=$ 

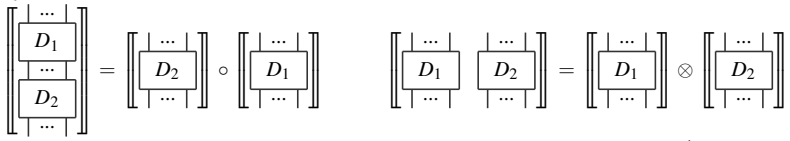
$D_1 \otimes D_2 :=$ 

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.

2.2 Standard Interpretation

In the *standard interpretation* [CD11], wires are interpreted with the two-dimensional Hilbert space, with orthonormal basis written as $\{|0\rangle, |1\rangle\}$, in Dirac notation [NC02]. Vectors of the form $|.\rangle$ (called "*kets*") are considered as columns vector, and therefore $|0\rangle = \binom{1}{0}$, $|1\rangle = \binom{0}{1}$, and $\alpha|0\rangle + \beta|1\rangle = \binom{\alpha}{\beta}$. Horizontal juxtaposition of wires is interpreted with the *Kronecker*, or *tensor* product. The tensor product of spaces $\mathscr{V}$ and $\mathscr{W}$ 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 $\begin{pmatrix}\alpha & \beta\end{pmatrix}$. 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: $\langle\phi \,|\, \psi\rangle$, 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 $\begin{pmatrix}1&0\\0&1\end{pmatrix} = \begin{pmatrix}1&0\\0&0\end{pmatrix} + \begin{pmatrix}0&0\\0&1\end{pmatrix} = \binom{1}{0}\begin{pmatrix}1&0\end{pmatrix} + \binom{0}{1}\begin{pmatrix}0&1\end{pmatrix} = |0\rangle\langle 0| + |1\rangle\langle 1|$. For more information on how Hilbert spaces, tensors, compositions and bras and kets work, we invite the reader to consult e.g. [NC02].

We are now ready to define the *standard interpretation*, where a diagram D with n inputs and m has for interpretation a map $[\![D]\!] : \mathbb{C}^{2^n} \to \mathbb{C}^{2^m}$. It is defined inductively as follows.



$$[\![\,|\,]\!] = id_{\mathbb{C}^2} = |0\rangle\langle 0| + |1\rangle\langle 1| \qquad [\![\times]\!] = \sum_{i,j\in\{0,1\}} |ji\rangle\langle ij| \qquad [\![\cap]\!] = [\![\cup]\!]^\dagger = |00\rangle + |11\rangle$$

$$[\![\text{H}]\!] = |+\rangle\langle 0| + |-\rangle\langle 1| \qquad [\![\alpha : n \to m]\!] = |0^m\rangle\langle 0^n| + e^{i\alpha}|1^m\rangle\langle 1^n|$$

2.3 Properties and structure

We list several definitions and known results of ZX-Calculus. See e.g. [Vil19] for more information.

Universality. ZX-diagrams are *universal* in the sense that for any linear map $f : n \to m$, there exists a diagram *connected* 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 [Bac14, JPV18].

Completeness. The ability to transform a diagram D_1 into a diagram D_2 using the rules of some axiomatization zx is denoted $\text{zx} \vdash D_1 = D_2$. The axiomatization is said to be *complete* whenever any two diagrams representing the same operator are axiomatically equivalent. Formally: $[\![D_1]\!] = [\![D_2]\!] \iff \text{zx} \vdash D_1 = D_2$

3 A Token Machine for ZX-diagrams

Inspired by the Geometry of Interaction [Gir89] and the associated notion of token machine [DR99] for proof nets [Gir96], 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 green node 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 1 (Tokens and Token States)**.** *Let D be a ZX-diagram. A* token *in D is a triplet $(e,d,b) \in \mathscr{E}(D) \times \{\downarrow,\uparrow\} \times \{0,1\}$. 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.

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 4.

Definition 2 (Asynchronous Evolution)**.** *Token states on a diagram D are equipped with two transition systems: (1) a* collision system *($\rightsquigarrow_c$), whose effect is to annihilate tokens; (2) a* diffusion sub-system *($\rightsquigarrow_d$), defining the flow of tokens within*

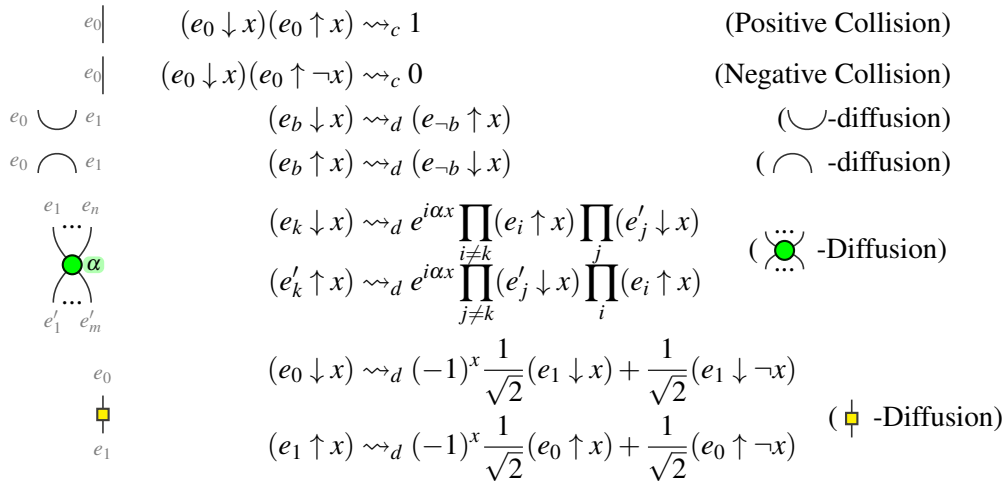


Table 1: Asynchronous token-state evolution, for all $x, b \in \{0,1\}$

D. In Table 1, 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.

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 one can always rewrite a token inside a cycle and get a non-terminating run or end up having two tokens on the same edge that do not collide.

We therefore set a rewriting strategy as follow:

Definition 3 (Collision-Free)**.** *A token state s of $\mathbf{tkS}(D)$ is called* collision-free *if whenever $s' \in \mathbf{tkS}(D)$, $s \not\rightsquigarrow_c s'$*

Definition 4 (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, $t \rightsquigarrow u$ iff $\exists t', t \rightsquigarrow_d t' \rightsquigarrow_c^* u$ and u is collision-free.*

3.2 Strong Normalization and Confluence

The token machine Rewrite System of Definition 4 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 the paper, 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. Also, with a notion of *cycle-balanceness* characterizing the behavior of tokens inside a cycle in the graph, $\rightsquigarrow$ terminates. We do not give the details in this extended-abstract.

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

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

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 5 can be made more general by considering multiple input tokens. 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 6 (Arbitrary Wire Initialisation)**.** *Let D be a connected ZX-diagram, with $\mathscr{I}(D) = [a_i]_{1\leq i\leq n}$, $\mathscr{O}(D) = [b_i]_{1\leq i\leq m}$, and $e \in \mathscr{E}(D) \neq \emptyset$ such that $(e\downarrow x)(e\uparrow x) \rightsquigarrow^* t_x$ for $x \in \{0,1\}$ with t_x a normal form. Then:*

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

4 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 token machine defined in [LFVY16] where some form of synchronicity is enforced.

The presented token machine works for *pure* quantum processes, i.e without any interaction with the environment. Since our machine is very general, a easy and natural extension to *mixed* processes is possible, to represent measurements. A wave-style semantic as also been studied.

We plan to further extend this token machine to handle recursion, something that doesn't exists in a operational form in the ZX-Calculus. We also plan to look at ways to detect if a diagram is unitary and to strengthen the relationship between Quantum Computation and Linear Logic.

References

[Amy18] M. Amy. Towards large-scale functional verification of universal quantum circuits. In *QPL'18*, volume 287 of *EPTCS*, pages 1–21, 2018.

[Bac14] M. Backens. The ZX-calculus is complete for stabilizer quantum mechanics. *New Journal of Physics*, 16(9):093021, 2014.

[CBB+] C. Chareton, S. Bardin *et al.* A deductive verification framework for circuit-building quantum programs. In *ESOP'21*, volume 12648 of *LNCS*, pages 148–177, 2021.

[CD11] B. Coecke and R. Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.

[dBH20] N. de Beaudrap and D. Horsman. The ZX calculus is a language for surface code lattice surgery. *Quantum*, 4:218, January 2020.

[DKPVDW20] R. Duncan, A. Kissinger *et al.* Graph-theoretic simplification of quantum circuits with the ZX-calculus. *Quantum*, 4:279, 2020.

[DR99] V. Danos and L. Regnier. Reversible, irreversible and optimal λ-machines. *Theoretical Computer Science*, 227(1-2):79–97, 1999.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

[Gir89] J.-Y. 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.

[Gir96] J.-Y. Girard. Proof-nets: the parallel syntax for proof-theory. *Lecture Notes in Pure and Applied Mathematics*, pages 97–124, 1996.

[Hil11] A. Hillebrand. *Quantum protocols involving multiparticle entanglement and their representations*. Master's thesis, University of Oxford, 2011.

[JPV18] E. Jeandel, S. Perdrix *et al.* A complete axiomatisation of the ZX-calculus for Clifford+T quantum mechanics. In *LICS'18*, pages 559–568, 2018.

[LFVY16] U. Dal Lago, C. Faggian *et al.* The geometry of parallelism. classical, probabilistic, and quantum effects. In *POPL'18*, pages 833—845, 2018.

[NC02] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2002.

[Vil19] R. Vilmart. *ZX-Calculi for Quantum Computing and their Completeness*. Ph.D thesis, Université de Lorraine, September 2019.