A Calculus for Topological Quantum Computation

Alessandra Di Pierro   Federica Panarotto
Dipartimento di Informatica, Università di Verona,
Strada le Grazie, 15 – 37134 Verona, Italy
alessandra.dipierro@univr.it   federica.panarotto@univr.it

Recent developments in theoretical physics have highlighted interesting topological features of some particular two-dimensional entities called anyons. Kitaev suggested that these features can be used to realise robust quantum computation, thus introducing the paradigm of topological quantum computation (TQC). The mathematics and physics of anyons is currently subject to intense investigations in all areas related to the study of quantum computation from both the foundational and the implementation viewpoint. In this paper we take a computer science viewpoint of TQC by presenting a quantum calculus whose terms are anyons and operations are suitable representations of the braid group on the terms.

1 Introduction

Topological quantum computation (TQC) is the name given to an alternative approach to quantum computation [8] where the elementary computational units are so-called *anyons* rather than *qubits*. It is called 'topological' because of the particular properties of anyons, that are physically realisable as quasiparticles in topological systems [11, 9]. More precisely, anyons behave differently from the more familiar three-dimensional particles (such as electrons, photons etc.). Their main features (and difference from qubits) is that they live in a two-dimensional space, and their 'charge' cannot be changed by local interactions. This is partly why they were suggested as a base for fault-tolerant quantum computation [5]. In this work we will abstract from the physical description of anyons as quasiparticles and consider them as classical fundamental particles with an internal quantum dimension. The quantum dimension of an anyon is an important notion for our treatment as it is used to calculate the dimension of the computational space. Its intuitive meaning is the asymptotic degeneracy of the ground state for that particle, and is in general very hard to calculate. For the simplest kind of anyons, that is the Fibonacci anyons, the quantum dimension grows like the Fibonacci numbers. The evolution of an anyon system occurs by braiding the particle trajectories according to rules which specify how pairs (or bipartite subsystems) behave under exchange. The mathematics of anyons and the relations with braids, topology and modular tensor categories is discussed in [11, 10]. In [7] Kauffman gived showed and efficient solution using anyons for the algorithm for the Jones polynomial problem. A discussion on the complexity of this algorithm is presented in [1, 6]. Here we are going to present an operational semantics for TQC which enlightens the programming language aspects of this paradigm. More precisely, we propose an encoding of anyons systems and their evolution in a calculus whose terms are made of anyons and computational steps corresponds to braiding of anyons. This will provide a general setting where important questions can be addressed and formally studied which are related to the universal computer and to the effective computational power of TQC (e.g. do anyons allow for more powerful algorithmic techniques than the circuit models?).

Submitted to:
QAPL 2014

2 Anyons Systems

As already mentioned, we will look at anyons as the unit elements of our calculus, which derive their properties from the physics behind their realisation as quasiparticles. Each anyon has associated a type, i.e. a label specifying the possible values of the physical charges, which we denote by using an enumeration $a_1, a_2, \ldots, a_n$, or simply by $a, b, c, \ldots$. We call *Types* the set of all the anyon types and we denote by $x, y, \ldots$ a variable representing an anyon of any type, while we use the subscript i to denotes an anyon of type a_i. In the following we will often use the term 'anyon' to indicate its type.

2.1 Anyons composition

From two anyons we can generate a new one by applying the *fusion rules* $a \otimes b = N_{ab}^{c} c$, where the number $N_{ab}^{c} (\in \mathbb{N})$ indicates the different ways of fusing a and b into c. These rules give the charge of a composed particle in terms of its constituents, and determine the particular anyonic model. We can use them in the opposite direction in order to split c into a and b and obtain two anyons from one. In this case we refer to the rules as *splitting rules*.

An anyon type a for which $\sum_c N_{ab}^{c} > 1$, for every b is called *non-Abelian*. In other words, a non-abelian anyon is one for which the fusion with another anyon may result in anyons of more than one type. This property is essential for computation because it implies a quantum dimension greater than 1 and therefore allows us to construct non-trivial computational spaces, i.e. spaces of dimension $n \geq 1$ of ground states where to store and elaborate information.

Considering the dual splitting process, a non-abelian anyon can therefore have more than one splitting rule that applies to it, e.g. $a \otimes b = c$ and $e \otimes g = c$.

Definition 1. An anyon *tree* is a tree where every internal node has two children labelled by the two anyons resulting from a splitting rule applied to the anyon labelling the node.

We obtain different trees depending on

- the shape of the tree resulting from the choice the anyon x to which we apply the fusion rules;
- the type of the anyons in the tree, resulting from the fusion rule chosen at every step of the construction of the tree.

We call the anyons at the leaves of the tree *leaf anyons*, the anyon at the root *root anyon* and the remaining anyons *internal anyons*.

2.2 Splitting space

In physical terms a system of anyons consists of a closed oriented surface Σ with anyons of types $a_1, \ldots, a_m$ located at distinct points $p_1, \ldots, p_m$. In our setting, we look at these anyons as the leaves of a tree constructed according to the splitting rules of a given model, that is as the final configuration of ground states at the end of the splitting process. This is the configuration to which unitary transformations can now be applied in order to perform the desired elaboration of the information encoded in the anyonic system.

As we work with non-abelian anyons, the splitting process applied to c generates as many trees as the number of splitting rules for c. If we now consider the leaves of each trees and apply again the splitting rules we obtain a bigger anyon system that we can still graphically represent by a tree. Again there will be as many trees as the number of splitting rules for each leaf. These trees representing the same global charge, namely the root, are orthogonal to each other as they have different internal nodes. We use the orthogonal trees with the same shape as a base of our computational space.

In general, given n anyons in the plane, we can arrange them along the real axis of the plane. If we fuse them consecutively, we obtain the fusion tree in Figure (a). Alternatively, we can consider the Hermitian conjugate of the fusion operator, i.e. splitting, and interpret fusion trees as splitting of one anyon into many, as in Figure (b). We will consider the tree in Figure (b) as the standard form of the terms of the calculus we introduce in the next section. For non-abelian anyons, there is of course a combinatorial number of ways of fusing (splitting) n anyons into a given anyon (the root of the tree or total charge). This number corresponds to the dimension $D(n)$ of the fusion (splitting) space, that is the Hilbert space with orthogonal basis given by all the fusion paths over the fixed fusion tree.

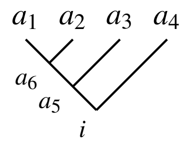


Figure 1: (a) Fusion Tree.

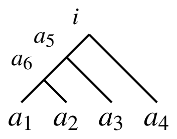


Figure 2: (b) Splitting Tree.

Definition 2 (Splitting space)**.** Given an anyon model that specifies the set *Types* and the fusion rules, the *splitting space* V_{split} is the set of all the anyon trees obtained by fixing the number m and types of the leaf anyons and the root anyon.

$$V_{\text{split}} := V^{x_0}_{x_1 \otimes x_2 \otimes x_3 \cdots \otimes x_m}, \text{ where } x_i \in \text{Types}.$$

We say that a tree is in *standard form* if the tree is constructed by selecting at each level i always the leftmost anyon as the splitting element x_i. The tree that we obtain is a (unbalanced) binary tree which grows only on the left. In the splitting space there are also other (non orthogonal) trees which have different shapes and contain only copies of information. This is because the total charge is conserved by locally exchanging two anyons. Thus, trees with different shapes which are obtained by applying the same splitting rules are ‘equivalent’ from a computational viewpoint, since they have the same information content.

3 A Calculus of Anyons

Computations with anyons is typically expressed in the languages of physics, abstract algebra or category theory. However, the typical approach to expressing computation in computer science is by means of programming languages. In this section we present a formal calculus for TQC which makes this new paradigm more amenable to investigations in the realm of theoretical computer science (such as the study of computability and complexity issues), and provides a more suitable base for the design of quantum programming languages.

We define our calculus in analogy with the classical Lambda calculus [2], i.e. as a calculus of terms with operations on them corresponding to the lambda-abstraction and function application.

3.1 The Language of terms

The following definition introduces the notion of *anyonic term*. Intuitively, an anyonic term represents an element in a subspace of the Hilbert space of trees associated to an anyon system. We assume a fixed set of types, $\mathcal{T} = \{a, b, \ldots\}$, or an enumeration of them $\mathcal{T} = \{a_1, a_2, \ldots, a_m\}$. We also assume an anyon system of n distinct particles and denote by H_n, the associated splitting space of dimension $D(n)$.

Definition 3 (Anyonic terms)**.** Anyonic terms are inductively defined by:

Simple terms For any type $a \in \mathcal{T}$, the trees with root a, in H_n are anyonic terms.

Abstraction If T is a simple term in H_n and x is a variable tree in H_k, with $k < n$, $T(x)$ defined as the tree T with a sub-tree x is an anyonic term.

Composition If T_1 and T_2 are simple terms, then $T_1 \cdot T_2$, defined as the tree with root a fusion result of the roots of T_1 and T_2, is an anyonic term.

Superposition Any vector in H_n (complex linear combination of simple terms) is an anyonic term.

Because of the tree equivalence mentioned before, we define the *computational space* of our calculus as the quotient space $\mathcal{S} = V_{\text{split}_{|\sim}}$, where $T_1 \sim T_2$ iff in the internal anyons of T_1 and T_2 have identical types in each position. We chose as a representative of each equivalence class the tree in standard form in that class.

A *computational state* (state) s is a superposition of classes, i.e. vectors, with amplitudes $\alpha_1, \cdots, \alpha_w \in \mathbb{C}$, i.e. $s = \alpha_1 v_1 \oplus \cdots \oplus \alpha_w v_w$.

3.2 Computing with anyons

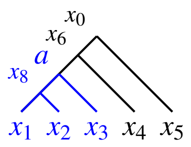
A computation on a system of non-Abelian anyons is performed by creating quasiparticles, *braiding* them and measuring their final result. Braiding can be seen as consisting in a twist or rotations of leaf anyons. Rotations on anyon trees are described by a braid group with the leaf anyons as objects and generators[1] σ_i. Thus we only need a finite set of generators $\{\sigma_i\}_{i<n}$ to construct all the rotations on terms. To manipulate the information stored in the trees in standard form we define a rotation σ_i as a matrix operator B_i that acts on the computational states. The B_i operations are the reductions of our calculus and they change the terms passing from one state to another. In fact, when we rotate two leaf anyons x_i and x_{i+1} we might change the type of the internal anyon x_j in some of their trajectories, thus transforming the original state (anyon tree) in a new one. We can however guarantee that this doesn't bring us outside the computational space.

Proposition 1. The operators $B_i \colon \mathcal{S} \to \mathcal{S}$ transform computational states (classes) in computational states.

The operators B_i represents the computational steps of our calculus and therefore the core of the operational semantics we are going to define. Informally, it consists in *function application*: Given an abstraction term $M(x)$ and a term N, the application of $M(x)$ to N is obtained in three steps: (1) 'binding' x to N, (2) 'evaluate' M after the instantiation, (3) 'read' the result. In the classical lambda-calculus this is called the β-reduction, and is the base of its operational semantics (in any of the different forms it is defined) [3]. In order to define an operational semantics for our anyonic calculus we therefore only need to specify (and formally define) what 'binding', 'evaluate' and 'read' means when we work with anyonic terms. We will only give here an informal explanation of our semantics and we refer to the full paper for the formal treatment.

Given an abstraction term $M(x)$ (as in the picture below), we denote its type by $b \to a$ indicating the subset inside the computational space for M where every trees in standard form has the internal anyon in position k of type a. The sub-tree (in blue) is constrained to have a root of type a in order to correctly replace the anyon x_7 of type a.

[1]A generator σ_i of braid group rotates the ith with the $i+1$th strand of a braid.



Suppose that we apply $M(x)$ to a term N of type a. Binding means that we instantiate the subtree rooted in x_7 in M with N. We can now apply the braiding, i.e. sequence of computational steps, to the instantiated tree $M(N)$. Intuitively this braiding represents the procedure M and realises the evaluation step. Finally, we read the result, which in quantum computation essentially means measuring the final state. We can obtain this by applying the same fusion rules used for constructing the term $M(N)$ in the reverse order and then extract the result from the resulting phase.

4 An Example

We consider a simple example of computation in the Fibonacci anyon model that we borrowed from [4], and we show how this can be implemented in our calculus. The following operator represents a controlled-phase gate, C-iP, on two quits:

$$\begin{bmatrix} e^{i\alpha} & 0 & 0 & 0 \\ 0 & e^{i\alpha} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-i\alpha} \end{bmatrix},$$

expressed with respect to the basis $\{00,01,10,11\}$. If we take the first qubit as the control qubit and the second as the target qubit, this gate for $\alpha = \pi$ corresponds to the controlled-iZ operation, where Z is the Pauli matrix

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

In our calculus this can be expressed as an abstraction term representing a function of the sub-term corresponding to the first qubit. Depending on this latter being 1 or 0, the iZ operation is applied or not to the term corresponding to the second qubit. The overal computation is depicted in Fig. 1.

The initial term is a composition of two terms representing a variable x and a simple term N, respectively, and both rooted in τ. Composition is obtained by applying to the root the fusion rule $\tau \otimes \tau = 1$. The term N is a simple term that simulates the qubit 0, while the variable x of type τ can be 1, 0 or any superposition.

We express the C-iZ gate as a function of the variable x (abstraction term). The sequence of basic steps implementing this operation can be obtained by means of the following distinct sequences (cf. [4]):

$$\begin{aligned} \text{Controller} &= B_2 * B_3^4 * B_2^2 * B_3^{-4} * B_2^2 * B_3^2 * B_2^2 * B_3^6 * B_2^2 * B_3^{-2} * \\ &\quad B_2^2 * B_3^{-2} * B_2^4 * B_3^{-4} * B_2^4 * B_3^{-2} * B_2^{-2} * B_3 \\ i - P &= B_4^{-2} * B_3^2 * B_4^4 * B_3^4 * B_4^2 * B_3^4 * B_4^4 * B_3^2 * B_4^{-2} * B_3^2 * B_4^4 * B_3^4 * B_4^2 * B_3^4 * B_4^4. \end{aligned}$$

For the reading process (i.e. the final measurement) we use the fusion space, whose states are duals to the states of the splitting space and are represented by inverse trees. We draw these trees at the end to close the computation and bring the system back to the vacuum (i.e. the anyon type 1. We now project

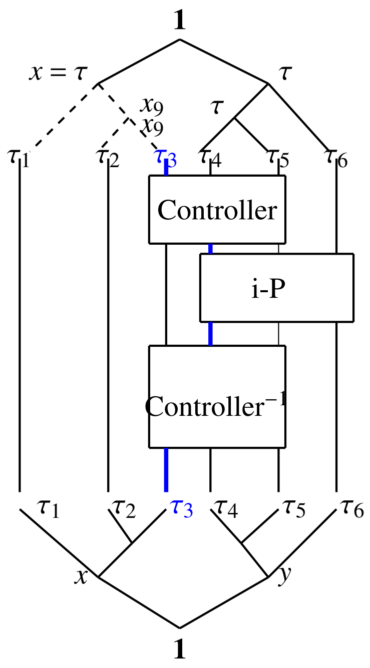


Figure 1: C-*iP* operator for a generic phase α.

the state after the computation using the dual states of the fusion space, in order to get the amplitudes associated with it.

We now show the numerical results that we obtain by applying the computation in Fig. 1 to the (anyonic encoding of the) initial state $|x1\rangle$.

We first apply the matrix corresponding to the function to the input state:

$$\begin{bmatrix} e^{i\alpha} & 0 & 0 & 0 \\ 0 & e^{i\alpha} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-i\alpha} \end{bmatrix} * |x1\rangle = e^{i\alpha}|01\rangle + e^{-i\alpha}|01\rangle,$$

and calculate the total probability as:

$$\langle\psi|C-iP|\psi\rangle = tr(C-iP) = e^{i\alpha}+1,$$

where $|\psi\rangle = |00\rangle + |01\rangle + |10\rangle + |11\rangle$. In order to read the result we project the final state into the dual state and divide it by the total probability:

$$\frac{\langle 01|}{\langle\psi|C-iP|\psi\rangle}(e^{i\alpha}|01\rangle + e^{i\alpha}|11\rangle) = \frac{e^{i\alpha}\langle 01|01\rangle}{e^{i\alpha}+1} = \frac{e^{i\alpha}}{e^{i\alpha}+1},$$

$$\frac{\langle 11|}{\langle\psi|C-iP|\psi\rangle}(e^{i\alpha}|01\rangle + e^{i\alpha}|11\rangle) = \frac{e^{i\alpha}\langle 11|11\rangle}{e^{i\alpha}+1} = \frac{e^{-i\alpha}}{e^{i\alpha}+1},$$

According to the function definition, we obtain the change of phase of $e^{-i\alpha}$ for the state $|11\rangle$, while for the state $|01\rangle$ we only get the overall phase produced by the C-iP operator.

References

[1] Dorit Aharonov, Vaughan Jones & Zeph Landau (2006): *A polynomial quantum algorithm for approximating the Jones polynomial*. In: *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, STOC '06, ACM, New York, NY, USA, pp. 427–436, doi:http://doi.acm.org/10.1145/1132516.1132579. Available at `http://doi.acm.org/10.1145/1132516.1132579`.

[2] H. P. Barendregt (1991): *The Lambda Calculus*, revised edition. *Studies in Logic and the Foundations of Mathematics* 103, North-Holland, Amsterdam.

[3] Hendrik Pieter Barendregt (1984): *The Lambda Calculus – Its Syntax and Semantics*. *Studies in Logic and the Foundations of Mathematics* 103, North-Holland.

[4] Layla Hormozi, Georgios Zikos, Nicholas E Bonesteel & Steven H Simon (2007): *Topological quantum compiling*. *Physical Review B* 75(16), p. 165310. Available at `http://prb.aps.org/abstract/PRB/v75/i16/e165310`.

[5] Alexei Kitaev (1997): *Fault-tolerant quantum computation by anyons*. *Annals of Physics* 303(1), p. 27. Available at `http://arxiv.org/abs/quant-ph/9707021`.

[6] Greg Kuperberg (2009): *How hard is it to approximate the Jones Polynomial?* *CoRR* abs/0908.0512. Available at `http://arxiv.org/abs/0908.0512`.

[7] Louis & H. Kauffman (1987): *State models and the Jones polynomial*. *Topology* 26(3), pp. 395 – 407, doi:10.1016/0040-9383(87)90009-7. Available at `http://www.sciencedirect.com/science/article/pii/0040938387900097`.

[8] M.A. Nielsen & I.L. Chuang (2000): *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, UK.

[9] Jiannis K. Pachos (2012): *Introduction to Topological Quantum Computation*. Cambridge University Press.

[10] Prakash Panangaden & Éric Oliver Paquette (2011): *A Categorical Presentation of Quantum Computation with Anyons*. In Bob Coecke, editor: *New Structures for Physics*, *Lecture Notes in Physics* 813, Springer Berlin / Heidelberg, pp. 983–1025. Available at `http://dx.doi.org/10.1007/978-3-642-12821-9_15`. 10.1007/978-3-642-12821-915.

[11] Zhenghan Wang (2010): *Topological Quantum Computation*. American Mathematical Soc.