

# Implementing and Automating complex arithmetic in quantomatic

Benjamin Frot



Thesis submitted to The University of Oxford  
for the degree of Master of Science

September 2011

## Abstract

Much work has been done on trying to understand and classify multipartite entanglement. For the past few years *Categorical Quantum Mechanics* [AC04], which introduces a formalism of quantum mechanics in terms of Symmetric Monoidal Categories (SMCs), has been used successfully in describing quantum systems at a high level of abstraction. From this work emerges a graphical calculus which is powerful enough to express any  $N$ -qubit entangled state and suitable for automated reasoning [CK10]. In [CKMR11] the authors show that it is possible to encode rational arithmetic within the GHZ/W-calculus.

In this dissertation, we propose an extension of the encoding of rational arithmetic to complex numbers and show how it is possible to approximate the complex field with arbitrary precision. We also present the work done on `quantomatic` [DD<sup>+</sup>] in order to support multiple theories. Finally, we show how both rational and complex arithmetics were implemented in this software.

## Acknowledgements

I would like to thank my supervisor, Bob Coecke, for his guidance throughout the research and writing phases of this dissertation. His feedback and numerous emails before the submission for QPL were very helpful. I am also grateful to Aleks Kissinger, Alex Merry and Lucas Dixon, all of whom answered countless questions. Their help with both `Quantomatic` and theoretical matters as well as their patience were invaluable.

I would like to thank Ron Nudel for the time spent proofreading early drafts of this dissertation.

For all the fun and edifying times spent with the boat club, during the Foundations of Physics meetings, at the talks and at the bar I would like to thank Wolfson college.

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>Background</b>	<b>8</b>
2.1	Categorical Quantum Mechanics . . . . .	8
2.1.1	Category Theory . . . . .	8
2.1.2	Quantum Computer Science . . . . .	8
2.2	Graphical Calculus . . . . .	10
2.3	Commutative Frobenius Algebras . . . . .	12
2.4	Multipartite Entanglement . . . . .	15
2.5	The GHZ/W Calculus . . . . .	16
2.5.1	Speciality - Antispeciality . . . . .	16
2.5.2	GHZ/W Pairs . . . . .	18
2.5.3	Plugging . . . . .	19
2.5.4	Phases . . . . .	19
<b>3</b>	<b>Rational Arithmetic in the GHZ/W Calculus</b>	<b>20</b>
3.1	Properties of GHZ phases . . . . .	20
3.2	Multiplicative inverses . . . . .	22
3.3	Additive inverses . . . . .	22
3.4	Conclusion . . . . .	23
<b>4</b>	<b>Complex Arithmetic</b>	<b>24</b>
4.1	$i$ . . . . .	24
4.2	Additive Inverses . . . . .	25
4.3	Encoding . . . . .	25
4.4	Conclusion . . . . .	28
<b>5</b>	<b>From Open Graphs to Quantomatic</b>	<b>29</b>
5.1	An Intuitive Approach . . . . .	29
5.2	Open Graphs . . . . .	30
5.3	Matching . . . . .	32
5.4	Rewriting . . . . .	33
5.5	Quantomatic . . . . .	34
5.5.1	Rewrite Rule . . . . .	36

5.5.2	Matching . . . . .	37
5.5.3	Rewriting . . . . .	37
5.5.4	Automation . . . . .	38
<b>6</b>	<b>Implementation in Quantomatic</b>	<b>40</b>
6.1	Coding . . . . .	40
6.1.1	Description of the software . . . . .	40
6.1.2	Support for multiple theories . . . . .	43
6.1.3	Perspectives . . . . .	44
6.2	Rewrite Rules for complex and rational arithmetic . . . . .	46
6.2.1	Rational arithmetic . . . . .	46
6.2.2	Complex Arithmetic . . . . .	46
6.3	Implementation and Results . . . . .	47
<b>7</b>	<b>Conclusion</b>	<b>50</b>
<b>A</b>	<b>Proof for 4.3</b>	<b>55</b>
<b>B</b>	<b>Code Listing</b>	<b>56</b>

# Chapter 1

## Introduction

It can be said that one of the main conceptual differences between Quantum Computer Science and Classical Computer Science lies in the fact that a Quantum Bit (qubit) can be in a superposition of the form  $\alpha_1|0\rangle + \alpha_2|1\rangle$ ,  $(\alpha_1, \alpha_2) \in \mathbb{C}^2$ . The construction of complex systems built from a combination of qubits gives access to a world which is still inaccessible to classical computing devices [BBC<sup>+</sup>93] [Ben92] [Sho97]. The foundations of Quantum Mechanics were laid in the early 1930's by von Neumann [vN96], who provided a robust framework based on Hilbert spaces in order to describe the behaviour of quantum systems. However, if one describes a qubit using  $\mathbb{C}^2$  and composes systems using the tensor product  $\otimes$ , the complexity of the whole system will increase exponentially with the number of qubits and in spite of the improvements brought forth by Dirac's notation, obtaining a nice, structural view of the system remains a difficult task. Thus, the Hilbert space formalism is perfect for performing operations on qubits since, in any case, it all boils down to multiplying matrices. Unfortunately, it makes it hard to reason about the system, to have a semantic approach to Quantum Mechanics.

*Categorical Quantum Mechanics* [AC04] aims, among other things, at solving this problem. Within this framework it is possible to develop *graphical languages* describing quantum processes and even to *automate* the reasoning thanks to graph rewriting.

In this dissertation we will focus on one of the applications of this framework: the encoding and automating of rational arithmetic within a graphical calculus. The reader can benefit from reading this work with either of two approaches in mind :

- The reader who is already familiar with Categorical Quantum mechanics and its applications will probably be interested in learning how expressive and powerful the GHZ/W-calculus is and how the tools implementing automated reasoning have evolved over the past few months.

- The reader who has a basic knowledge of Category Theory can see this dissertation as a means of getting acquainted with Categorical Quantum Mechanics through an example.

The contributions of this thesis are the following:

- We extend the encoding of rational arithmetic within the GHZ/W-calculus to the field of complex numbers.
- We propose a set of rewrite rules which allow the automation of this encoding.
- We present in this thesis the work undertaken to achieve a full support of arbitrary theories in `quantomatic` [DD<sup>+</sup>].
- We implement rational arithmetic in `quantomatic` : core theory and rewrite rules.
- We implement complex arithmetic in `quantomatic` by implementing another theory and the corresponding rewrite rules.
- We suggest the implementation of plug-ins in `quantomatic` and give a proof of concept.

In the next section, which assumes familiarity with basic notions about categories, we give some necessary background. Then, in chapter 3, we recall the theory behind encoding of rational arithmetic in the GHZ/W-calculus. In chapter 4 we propose an extension from rational arithmetic to complex arithmetic. Chapters 5 and 6 are dedicated to `Quantomatic` : the theory of Open Graphs behind it and the actual implementation of both rational and complex arithmetic.

# Chapter 2

## Background

As stated in the introduction, this dissertation deals with the Categorical interpretation of quantum mechanics [AC04] and, specifically, the graphical calculi that this framework allows to define. We give here the references and definitions required to fully understand what follows in this dissertation.

### 2.1 Categorical Quantum Mechanics

#### 2.1.1 Category Theory

The reader is expected to be well acquainted with category theory and especially monoidal categories, a key concept in the graphical interpretation of quantum processes. The lecture notes on Category Theory referenced by [AT11] are a good starting point. [CP09] gives a perfect introduction for the physicist interested in Category Theory and Quantum Computer Science while a good book on this topic would be [Lan98].

#### 2.1.2 Quantum Computer Science

Familiarity with Quantum Computer Science and von Neumann's formalism is also important. Andreas Doering's lecture notes [Doe10] on the topic provide a good introduction to this formalism and to the important concepts of Quantum Computer Science. Furthermore, Categorical Quantum Mechanics [AC04] will be used more extensively. This framework describes quantum processes in terms of monoidal categories with additional structures. Most of the essential structures that one can depict in von Neumann's formalism can be expressed in terms of monoidal categories and it makes it possible to give a more abstract and yet structural picture of quantum processes. A good introduction to this topic can be found in [Coe05] and [Coe09]. In order to illustrate what can be done within this framework we show how states, as we know them in von Neumann's formalism, can be expressed.



First, let us recall a few important definitions and theorems.

**Definition** A *monoidal category*  $(\mathcal{C}, \otimes, I, \alpha, \lambda, \rho)$  is a category  $\mathcal{C}$  with a bifunctor  $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ , an object  $I \in \mathcal{C}$  and three natural isomorphisms  $\alpha, \lambda$  and  $\rho$

$$\begin{aligned}\alpha_{A,B,C} : A \times (B \times C) &\xrightarrow{\cong} (A \times B) \times C \\ \lambda_A : I \times A &\xrightarrow{\cong} A \\ \rho_A : A \times I &\xrightarrow{\cong} A\end{aligned}$$

such that  $\lambda_I = \rho_I : I \times I \rightarrow I$  and that the following diagrams commute:

$$\begin{array}{ccc} A \times (B \times (C \times D)) & \xrightarrow{\alpha} & (A \times B) \times (C \times D) \xrightarrow{\alpha} ((A \times B) \times C) \times D \\ \downarrow id \times \alpha & & \alpha \times id \uparrow \\ A \times ((B \times C) \times D) & \xrightarrow{\alpha} & (A \times (B \times C)) \times D \\ & & \uparrow \\ & & A \times (I \times B) \xrightarrow{\alpha} (A \times I) \times B \\ & & \downarrow id \times \lambda \quad \swarrow \rho \times id \\ & & A \times B \end{array}$$

**Definition** A *symmetric monoidal category* (SMC) is a monoidal category  $(\mathcal{C}, \otimes, I, \alpha, \lambda, \rho)$  together with an additional natural isomorphism

$$\sigma_{A,B} : A \otimes B \xrightarrow{\cong} B \otimes A$$

such that for all  $A, B$  and  $C$ :

$$\sigma_{B,A} \circ \sigma_{A,B} = id_{A \otimes B}, \quad \lambda_A \circ \sigma_{A,I} = \rho_A .$$

**Example 2.1.1.** If we call **FdHilb** the category of finite Hilbert spaces and linear maps, then  $(\mathbf{FdHilb}, \otimes, \mathbb{C})$  is an SMC.

The tutorial chapters of [CD11] contain further information on the notions that we give hereafter.

**Definition** [AC05] A  $\dagger$ -symmetric monoidal category ( $\dagger$ -SMC) is a symmetric monoidal category equipped with an identity-on-objects contravariant endofunctor

$$(-)^\dagger : \mathcal{C}^{op} \rightarrow \mathcal{C}$$

which assigns to each morphism  $f : A \rightarrow B$  an *adjoint* morphism  $f^\dagger : B \rightarrow A$  which coherently preserves the monoidal structure, that is:

$$(f \circ g)^\dagger = g^\dagger \circ f^\dagger, \quad (f \otimes g)^\dagger = f^\dagger \otimes g^\dagger, \quad 1_A^\dagger = 1_A, \quad f^{\dagger\dagger} = f.$$

**Example 2.1.2.** ***FdHilb** is a  $\dagger$ -SMC,  $(-)^{\dagger}$  being the functor which associates any linear map with its adjoint as we know them in linear algebra.*

More information about the specific properties of compact closed categories can be found in [KL80].

**Definition** The *points* of  $A$  in a category  $\mathcal{C}$  are the morphisms of the type  $\psi : I \rightarrow A$ .

**Definition** The *scalars* of  $\mathcal{C}$  are the points of  $I$  in  $\mathcal{C}$ .

**Example 2.1.3.** *Any linear map  $\psi : \mathbb{C} \rightarrow \mathcal{H}$ ,  $\mathcal{H} \in \text{Obj}(\mathbf{FdHilb})$  is uniquely determined by  $\psi(1)$ . It is then possible to express states as we know them using the points defined above. Similarly, scalars being just a special kind of vector, we see that any scalar  $c \in \mathbb{C}$  can be expressed as a scalar in the sense of the previous definition.*

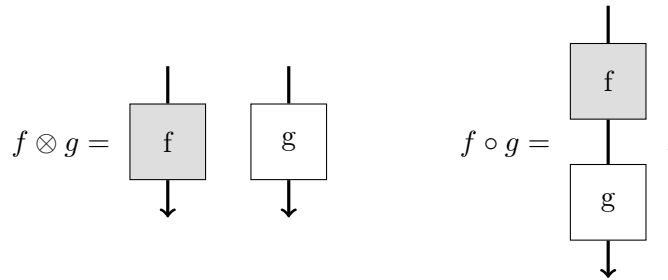
It is possible to define a scalar multiplication and to prove that points and scalars behave as expected in **FdHilb** regarding distributivity and other noteworthy properties [CD11]. Let us also remark that **FdHilb** is not the only interesting category which can be equipped with the required structures. The category **FRel** of finite sets and binary relations and its subcategory **Spek** [Spe07] [CE11b] are also rich enough to simulate protocols as important as quantum teleportation.

## 2.2 Graphical Calculus

We saw in the previous section that two kinds of composition can be observed in SMCs. The composition law  $\circ$  is inherent to categories and the tensor product  $\otimes$  exists as such because those categories are monoidal. Thus, morphisms can be composed along two different dimensions, one corresponding to  $\circ$  and another one corresponding to  $\otimes$ . One can see the composition of morphisms as a temporal dimension whereas the tensor product corresponds to a spatial dimension. Using the tensor diagram notation introduced by Penrose [Pen71], the author of [Sel07] introduces a graphical notation for morphisms in Compact Closed Categories. For a review of graphical languages defined in monoidal categories see [Sel09] by the same author.

Objects are represented as edges labelled with the names of the corresponding objects and arrows as boxes labelled by the names of the corresponding morphisms. When the domains and codomains of the arrows

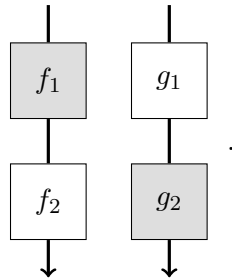
are obvious, edges are left unlabelled. Usual composition of morphisms is expressed by connecting outputs to inputs and tensor product is given by putting boxes side by side. The identity arrow is represented by a blank edge.



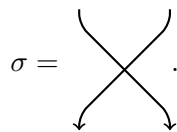
Using this notation, the well known, yet not trivial, property

$$(f_1 \circ f_2) \otimes (g_1 \circ g_2) = (f_1 \otimes g_1) \circ (f_2 \otimes g_2)$$

becomes obvious and is automatically embedded within the graphical calculus:



The symmetry map  $\sigma$  is represented by crossed wires:



Finally, bras and kets can also be captured by this graphical calculus. Define  $|\psi\rangle :: 1_A \mapsto |\psi\rangle$  and  $\langle\psi|$  can coherently be represented by:

$$|\psi\rangle = \begin{array}{c} \triangle \\ \uparrow \\ \psi \end{array} \quad \langle\psi| = \begin{array}{c} \downarrow \\ \psi \\ \triangle \end{array} .$$

The soundness of this graphical calculus was showed by Joyal and Street [JS91] and is stated in the following theorem:

**Theorem 2.2.1.** *Two maps consisting of arbitrary compositions and tensor products of smaller maps :  $f : \otimes A_i \rightarrow \otimes B_j$  and swap maps  $\sigma_{A,B}$  are equal if their graphical representations are equal.*

## 2.3 Commutative Frobenius Algebras

**Definition** Let  $\mathcal{C} = (\mathcal{C}, \otimes, I, \alpha, \lambda, \rho)$  be a monoidal category with monoidal product  $\otimes$  and unit  $I$ . A *monoid*  $M$  in  $\mathcal{C}$  is an object  $M \in \text{Ob}(\mathcal{C})$  together with two arrows  $\mu : M \otimes M \rightarrow M, \eta : I \rightarrow M$  such that the following diagrams commute:

$$\begin{array}{ccc}
 M \boxtimes (M \otimes M) & \xrightarrow{\alpha} & (M \otimes M) \otimes M \xrightarrow{\mu \otimes id} M \otimes M \\
 \downarrow id \otimes \mu & & \downarrow \mu \\
 M \otimes M & \xrightarrow{\mu} & M,
 \end{array}$$
  

$$\begin{array}{ccccc}
 e \otimes M & \xrightarrow{\eta \otimes id} & (M \otimes M) & \xleftarrow{id \otimes \eta} & M \otimes e \\
 & \searrow \lambda & \downarrow \mu & \swarrow \rho & \\
 & & M & & 
 \end{array}$$

Note that the definition of the monoid  $M = (M, \mu, \eta)$  makes use of the structures of the monoidal category  $\mathcal{C}$  (namely of the unit  $I$ , the associator  $\alpha$  and the isomorphisms  $\lambda$  and  $\rho$ ).

Moreover, if  $\mu = \mu \circ \sigma$  the monoid is said *commutative*.

In the graphical languages described in the previous section, a monoid  $(M, \mu, \eta)$  is represented by a triple  $(M, \text{multiplication}, \text{unit})$  and the previous conditions can be expressed graphically by

**Remark** A monoid in the monoidal category  $(\mathbf{Set}, \times, \{*\})$  is a monoid in the usual sense. Indeed, if  $(M, \mu, \eta)$  is a monoid in  $\mathbf{Set}$  it can easily be verified that  $(M, \mu, \eta(\{*\}))$  meets all the conditions required to be a monoid in the usual sense.

**Definition** A *comonoid* in a monoidal category  $\mathcal{C}$  is a monoid in the dual category  $\mathcal{C}^{op}$ . Graphically a comonoid is represented by the triple  $(M, \delta = \text{comultiplication}, \epsilon = \text{counit})$ .

Moreover, if  $\sigma \circ \delta = \delta$  the comonoid is said *cocommutative*.

**Remark** In  $\mathbf{FdHilb}$  comonoids are coalgebras.

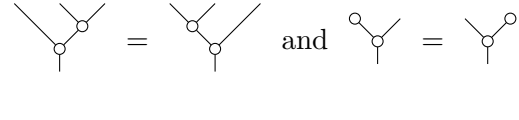
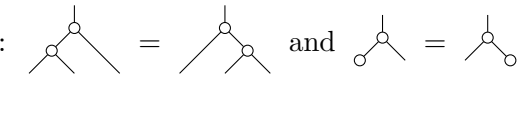
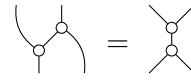
We can now define a structure which is of great importance for a categorical interpretation of quantum processes: Frobenius Algebras. Note that we give here the categorical definition of a Frobenius algebra. Hence, this is, more or less, a misuse of the word algebra.

**Definition** A *Frobenius Algebra*  $(A, \mu, \eta, \delta, \epsilon)$  in a monoidal category  $\mathcal{C}$  consists of a monoid  $(A, \mu, \eta)$  and a comonoid  $(A, \delta, \epsilon)$  such that the diagrams

$$\begin{array}{ccc} A \otimes A & \xrightarrow{\delta \otimes 1_A} & A \otimes A \otimes A \\ \downarrow \mu & & \downarrow 1_A \otimes \mu \\ A & \xrightarrow{\delta} & A \otimes A, \end{array}$$

$$\begin{array}{ccc} A \otimes A & \xrightarrow{1_A \otimes \delta} & A \otimes A \otimes A \\ \downarrow \mu & & \downarrow \mu \otimes 1_A \\ A & \xrightarrow{\delta} & A \otimes A, \end{array}$$

commute, or explicitly, that  $(\mu \otimes 1_A)(1_A \otimes \delta) = (1_A \otimes \mu)(\delta \otimes 1_A) = \delta\mu$ . Graphically, a Frobenius algebra depicted by  $(A, \text{multiplication}, \text{comultiplication}, \text{multiplication}, \text{comultiplication})$  verifies the identities

- $(A, \text{multiplication}, \text{comultiplication})$  is a monoid : 
- $(A, \text{comultiplication}, \text{comultiplication})$  is a comonoid : 
- Frobenius condition : 

If  $(A, \text{multiplication}, \text{comultiplication})$  and  $(A, \text{comultiplication}, \text{comultiplication})$  are respectively commutative and co-commutative, then the Frobenius Algebra is said *commutative*.

Hereafter Commutative Frobenius Algebras (CFAs) will be distinguished by the colour of their dots.

**Example 2.3.1.** Given an orthonormal basis  $\{|0\rangle, \dots, |d-1\rangle\}$  of  $\mathbb{C}^d$  the maps

$$\begin{aligned} \mu &= \sum_i |i\rangle\langle ii| : \mathbb{C}^d \otimes \mathbb{C}^d \rightarrow \mathbb{C}^d & \eta &= \sum_i |i\rangle : \mathbb{C} \rightarrow \mathbb{C}^d \\ \delta &= \sum_i |ii\rangle\langle i| : \mathbb{C}^d \rightarrow \mathbb{C}^d \otimes \mathbb{C}^d & \epsilon &= \sum_i \langle i| : \mathbb{C}^d \rightarrow \mathbb{C} \end{aligned}$$

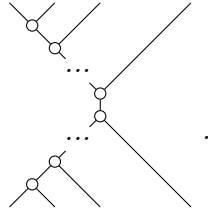
form a CFA on  $\mathbb{C}^d$ .

We will now give an important result regarding CFAs and a few useful pieces of notation that we will reuse in the next chapters.

**Definition** Given a CFA  $\mathcal{F} = (A, \mu, \eta, \delta, \epsilon)$  an  $\mathcal{F}$ -graph is a map obtained from the vertical and horizontal composition of  $\mu, \delta, \eta, \epsilon, \sigma$  and  $1_A$ . An  $\mathcal{F}$ -graph is said *connected* if its graphical representation is connected.

We will now state a weaker version of the theorem proved in [Koc03].

**Theorem 2.3.2.** *Any connected  $\mathcal{F}$ -graph with no loops admits the normal form*



When a graph is reduced to such a form it is called a *Spider*, which is defined by its number of inputs  $n$  and its number of outputs  $m$ .

$$S_m^n := \text{diagram of a spider with } n \text{ inputs and } m \text{ outputs}$$

When a spider has no input or output, the spiders  $S_m^0$  and  $S_0^n$  are defined by

$$S_m^0 := S_m^1 \circ \uparrow \quad S_0^n := \downarrow \circ S_1^n.$$

**Definition** The map  $S_2^0$  (resp.  $S_0^2$ ) is called a *cap* (resp. a *cup*).

**Proposition 2.3.3.** [CW87] *For any Frobenius Algebra  $(A, \mu, \eta, \delta, \epsilon)$*

$$\text{cap} = \text{cup} = \text{vertical line}$$

*holds. This is known as the compactness property. A short proof which makes use only of the conditions defining Frobenius Algebras can be found in [Her10].*

We will see at the end of this chapter how the notion of CFA can be used in quantum mechanics and we will use those definitions extensively throughout the rest of this dissertation.

## 2.4 Multipartite Entanglement

Entanglement is often described as the most powerful characteristic of quantum systems. Quantum teleportation and superdense coding are use cases of this property. In quantum computer science, entanglement has been used to develop algorithms that boost computational power significantly [Sho97]. It is often useful to arrange quantum states in classes which reflect their computational power. We give a definition of the most widely used classification and use it to show that the case in which  $n = 3$  qubits, two states, namely GHZ and W, stand out. The interested reader will find useful information in [DVC00].

**Definition** Two states are *LOCC-equivalent* if and only if they can be deterministically inter-converted with local physical operations and classical communication. (LOCC standing for Local Operations and Classical Communication.)

**Definition** Two states are *SLOCC-equivalent* if and only if they can be made LOCC-equivalent with a non-zero probability. SLOCC standing for Stochastic LOCC.

**Theorem 2.4.1.** *For an  $n$  qubit system, two states  $|\psi\rangle$  and  $|\phi\rangle$  are SLOCC-equivalent iff there exist invertible linear maps  $L_i : \mathbb{C}^2 \rightarrow \mathbb{C}^2$ ,  $i \in \{1, \dots, n\}$  such that*

$$|\psi\rangle = (L_1 \otimes \dots \otimes L_n)|\phi\rangle.$$

If we consider a system with three qubits  $A, B$  and  $C$  there are different classes of entanglements under SLOCC:

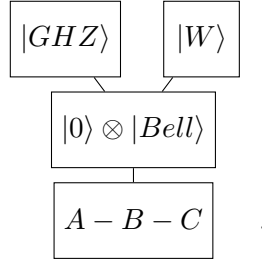
- $A - B - C$  : in this configuration the qubits are not entangled and the system can be reduced to  $|0\rangle \otimes |0\rangle \otimes |0\rangle$ .
- $A - BC$ ,  $AB - C$  or  $C - AB$  : two of the qubits are entangled and the system can be reduced to  $\frac{1}{\sqrt{2}}|0\rangle \otimes (|00\rangle + |11\rangle) = |0\rangle \otimes |Bell\rangle$ .
- $ABC$  : genuine three-qubit entanglement.

If a state  $|\psi\rangle$  can be converted under SLOCC into a state  $|\phi\rangle$  then we note  $|\phi\rangle \preceq |\psi\rangle$ . As suggested by this piece of notation,  $\preceq$  is a preorder relation and it defines the notion of a *maximally* entangled state.

In the case  $n = 3$ , which is one of interest for us, there are two maximally entangled states with respect to SLOCC:  $|GHZ\rangle$  and  $|W\rangle$  [DVC00]

$$\begin{aligned} |GHZ\rangle &= \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \\ |W\rangle &= \frac{1}{\sqrt{2}}(|001\rangle + |010\rangle + |100\rangle) \end{aligned}$$

and we can represent the poset of classes for three qubits:



A good intuition as to what these states are is given in [CE11a]. It seems that the  $|W\rangle$  expresses a kind of pairwise entanglement between the qubits, whereas the  $|GHZ\rangle$  state is globally shared between all the qubits.

## 2.5 The GHZ/W Calculus

Having introduced the GHZ and W states in the previous section and having given the main rules used for the graphical representation of quantum processes in  $\dagger$ -SMCs, it is now possible to introduce the graphical calculus that will be of specific interest in the next chapters : the GHZ/W Calculus.

### 2.5.1 Speciality - Antispeciality

As we will see in the next section, the GHZ and W states are closely related to the notion of *Speciality* and *Antispeciality*, which we define here.

**Definition** A CFA  $(A, \begin{array}{c} \diagup \\ \diagdown \end{array}, \begin{array}{c} \circ \\ | \end{array}, \begin{array}{c} | \\ \circ \end{array}, \begin{array}{c} \diagdown \\ \diagup \end{array}, \begin{array}{c} | \\ \circ \end{array})$  is called a *Special Commutative Frobenius Algebra* (SCFA) if

$$\begin{array}{c} \circ \\ | \\ \circ \\ | \end{array} = \left| \right.$$

**Definition** A CFA  $(A, \begin{array}{c} \diagup \\ \diagdown \end{array}, \begin{array}{c} \bullet \\ | \end{array}, \begin{array}{c} | \\ \bullet \end{array}, \begin{array}{c} \diagdown \\ \diagup \end{array}, \begin{array}{c} | \\ \bullet \end{array})$  is called an *Antispecial Commutative Frobenius Algebra* (ACFA) if

$$\circ \begin{array}{c} | \\ \bullet \\ | \end{array} = \begin{array}{c} | \\ \bullet \\ \circ \\ | \end{array}$$

where

$$\begin{array}{c} \bullet \\ | \\ \bullet \end{array} := \begin{array}{c} \bullet \\ | \\ \bullet \\ | \end{array} \quad \begin{array}{c} | \\ \bullet \end{array} := \begin{array}{c} | \\ \bullet \\ | \end{array}$$

In **FdHilb** the a direct link can be made between SCFAs (resp ACFA) on  $\mathbb{C}^2$  and the GHZ (resp the W) state.



**Theorem 2.5.1.** [CK10] For any SCFA on  $\mathbb{C}^2$ , the induced frobenius state is SLOCC-equivalent to  $|GHZ\rangle$ .

**Theorem 2.5.2.** [CK10] For any ACFA on  $\mathbb{C}^2$ , the induced frobenius state is SLOCC-equivalent to  $|W\rangle$ .

It is, then, interesting to see which CFAs induce the  $|GHZ\rangle$  and the  $|W\rangle$  states because any  $|GHZ\rangle$  (resp  $|W\rangle$ ) structure is a  $|GHZ\rangle$  (resp  $|W\rangle$ ) state.

A SCFA corresponding to a  $|GHZ\rangle$  structure is, for instance, defined by the four maps

$$\begin{aligned} \begin{array}{c} \diagup \\ \circ \\ \diagdown \end{array} &= |0\rangle\langle 00| + |1\rangle\langle 11| & \begin{array}{c} \circ \\ \diagup \\ \diagdown \end{array} &= |0\rangle + |1\rangle \\ \begin{array}{c} \diagdown \\ \circ \\ \diagup \end{array} &= |00\rangle\langle 0| + |11\rangle\langle 1| & \begin{array}{c} \circ \\ \diagdown \\ \diagup \end{array} &= \langle 0| + \langle 1|, \end{aligned}$$

whereas an example of a  $W$ -structure would be

$$\begin{aligned} \begin{array}{c} \bullet \\ \diagup \\ \diagdown \end{array} &= |1\rangle\langle 11| + |0\rangle\langle 01| + |0\rangle\langle 10| & \begin{array}{c} \bullet \\ \diagup \\ \diagdown \end{array} &= |1\rangle \\ \begin{array}{c} \bullet \\ \diagdown \\ \diagup \end{array} &= |00\rangle\langle 0| + |01\rangle\langle 1| + |10\rangle\langle 1| & \begin{array}{c} \bullet \\ \diagdown \\ \diagup \end{array} &= \langle 0|. \end{aligned}$$

Any CFA  $(A, \begin{array}{c} \diagup \\ \circ \\ \diagdown \end{array}, \begin{array}{c} \circ \\ \diagup \\ \diagdown \end{array}, \begin{array}{c} \diagdown \\ \circ \\ \diagup \end{array}, \begin{array}{c} \circ \\ \diagdown \\ \diagup \end{array})$  induces a tripartite state defined by the spider  $S_3^0$ . Thus, we can see easily that the  $GHZ$  and  $W$  structure given above induce the states

$$\begin{array}{c} \circ \\ \diagup \\ \diagdown \end{array} := \begin{array}{c} \circ \\ \diagup \\ \diagdown \end{array} \begin{array}{c} \circ \\ \diagup \\ \diagdown \end{array} = |000\rangle + |111\rangle = |GHZ\rangle^1$$

and

$$\begin{array}{c} \bullet \\ \diagup \\ \diagdown \end{array} := \begin{array}{c} \bullet \\ \diagup \\ \diagdown \end{array} \begin{array}{c} \bullet \\ \diagup \\ \diagdown \end{array} = |001\rangle + |010\rangle + |100\rangle = |W\rangle^2.$$

**Remark** In the previous example we give an equality between the graphical representation of a map and its corresponding expression in the Dirac notation. Obviously, this requires doing some (easy) calculations. Because these calculations are not of any real interest to the reader, we will never show them and shall give the result directly. However, a program written by the author made available in the appendix can be used to check the validity of the statements made in this dissertation. The corresponding command will be given as a footnote and will allow an easy verification of the identity.

<sup>1</sup>./g2m.py "otimes(deltaGHZ, id) \* deltaGHZ \* etaGHZ" "otimes(otimes(zero, zero), zero) + otimes(otimes(one, one), one)"

<sup>2</sup>./g2m.py "otimes(deltaW, id) \* deltaW \* etaW" "otimes(otimes(one, zero), zero) + otimes(otimes(zero, one), zero) + otimes(otimes(zero, zero), one)"



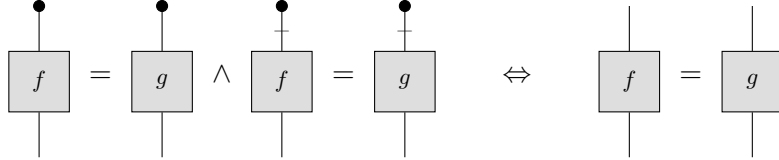
### 2.5.3 Plugging

When working in a category for which the objects are finite, it is possible to prove graphically properties using a technique called *plugging*. Combining this technique and the axioms of the GHZ/W calculus allows one to prove much more graphical identities.

In the case of a qubit, we consider the category **FdHilb** and GHZ/W pairs are defined over  $\mathcal{Q} = \mathbb{C}^2$ . In this particular configuration we have the following result:

**Lemma 2.5.5.** [CK10] For a GHZ/W pair on  $\mathcal{H}$  in **FdHilb** with  $\dim(\mathcal{H}) \geq 2$ , the points  $\bullet$ ,  $\dagger$  span a 2-dimensional space.

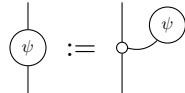
In the case  $\mathcal{H} = \mathcal{Q}$ , we conclude that they form a basis, and if two linear maps coincide on a basis then, they are equal. Consequently



holds.

### 2.5.4 Phases

**Definition** [CKMR11] For a given CFA  $(A, \text{Y}, \text{O}, \text{X}, \text{B})$ , a *phase*  $f : A \rightarrow A$  is a morphism of the form



for some element  $\psi : I \rightarrow A$ .

We will see in the next chapter how some properties of GHZ phases can be used in order to implement rational arithmetic.

**Example 2.5.6.** What are the GHZ phases in **FdHilb**? Let  $|\psi\rangle : \alpha|0\rangle + |1\rangle$  be the state of a qubit. We take  $|\psi|^2 = D$  which is 2 in the case of a qubit, and  $|\psi\rangle$  is of the form  $|0\rangle + e^{i\theta}|1\rangle$ ,  $\theta \in [0, 2\pi[$ . For more details as to why we choose to restrain ourselves to  $|\psi|^2 = D$  see [CD11]. Then phases are the maps  $\text{Y} \circ (1 \otimes \psi)$  and they admit the following family of matrices:

$$= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}.$$

## Chapter 3

# Rational Arithmetic in the GHZ/W Calculus

In this chapter we give a commented summary of *The GHZ/W-calculus contains rational arithmetic* [CKMR11]. Indeed, this paper sets the foundation upon which this is based and should be read by the interested reader. However, in an effort to make this thesis self-contained, we will give and explain the main results here, so that the next chapter about complex arithmetic can be understood without reading [CKMR11].

### 3.1 Properties of GHZ phases

We give here a few important theorems and corollaries carefully proved by plugging in [CKMR11].

**Theorem 3.1.1.**

As we shall see shortly, this theorem expresses, up to a scalar, a kind of distributivity law of the GHZ-phase over

**Theorem 3.1.2.**

For GHZ-phases let us define a very suggestive piece of notation.

**Theorem 3.1.3.**

$$\begin{array}{c} \psi \\ \circ \\ \hline \frac{1}{\psi} \\ \circ \end{array} = \begin{array}{c} \psi \quad \psi \\ \circ \quad \circ \\ \bullet \quad \bullet \end{array} \quad \Bigg|$$

The map  $\begin{array}{c} \diagup \\ \bullet \\ \diagdown \end{array}$  (resp  $\begin{array}{c} \diagdown \\ \bullet \\ \diagup \end{array}$ ) will be called *addition* (resp *multiplication*). Similarly,  $\begin{array}{c} \bullet \\ \diagup \\ \circ \end{array}$  (resp  $\begin{array}{c} \bullet \\ \diagdown \\ \circ \end{array}$ ) is call the *unit for addition* (resp *unit for multiplication*). Moreover, the following corollary shows that, up to a scalar, there is a *distributivity law* of multiplication over addition.

**Corollary 3.1.4.**

$$\begin{array}{c} \psi \\ \circ \\ \hline \bullet \end{array} \begin{array}{c} \Phi \quad \phi \\ \circ \quad \circ \\ \bullet \end{array} = \begin{array}{c} \psi \quad \Phi \quad \psi \quad \phi \\ \circ \quad \circ \quad \circ \quad \circ \\ \bullet \end{array}$$

In order to do arithmetic on  $\mathbb{N}$ , the following encoding of natural numbers is defined:

$$\begin{array}{c} 0 \\ \circ \\ \hline \bullet \end{array} = \begin{array}{c} \bullet \\ \hline \bullet \end{array} \quad \begin{array}{c} n+1 \\ \circ \\ \hline \bullet \end{array} = \begin{array}{c} n \quad \circ \\ \bullet \quad \bullet \\ \hline \bullet \end{array}$$

We can now verify that  $\begin{array}{c} \diagdown \\ \bullet \\ \diagup \end{array}$  behaves as expected and that addition and multiplication interact as in the case of natural numbers.

**Remark** When working in  $\mathbf{FHilb}_p$  invertible scalars can be ignored and we will use simplified versions of the previous theorems.

**Example 3.1.5.**  $\begin{array}{c} \diagdown \\ \bullet \\ \diagup \end{array}$  is the normal addition. (Using Th. 2.3.2)

$$\begin{array}{c} n \quad m \\ \bullet \quad \bullet \\ \hline \bullet \end{array} = \begin{array}{c} \overbrace{\circ \dots \circ}^n \quad \overbrace{\circ \dots \circ}^m \\ \bullet \quad \bullet \quad \bullet \\ \hline \bullet \end{array} = \begin{array}{c} \overbrace{\circ \dots \circ}^n \quad \overbrace{\circ \dots \circ}^m \\ \bullet \quad \bullet \\ \hline \bullet \end{array} = \begin{array}{c} m+n \\ \circ \\ \hline \bullet \end{array}$$

**Example 3.1.6.**  $\begin{array}{c} \diagdown \\ \bullet \\ \diagup \end{array}$  and  $\begin{array}{c} \diagdown \\ \circ \\ \diagup \end{array}$  interact like normal addition and normal multiplication.

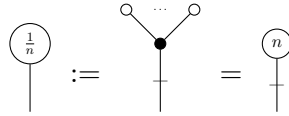
$$\begin{array}{c} a \quad b \quad c \\ \circ \quad \circ \quad \bullet \\ \hline \circ \end{array} \stackrel{3.1.1}{=} \begin{array}{c} a \quad b \quad a \quad c \\ \circ \quad \circ \quad \circ \quad \circ \\ \bullet \quad \bullet \\ \hline \bullet \end{array} \stackrel{3.1.1}{=} \begin{array}{c} \overbrace{a \quad \circ \quad \dots \quad \circ \quad a}^b \quad \overbrace{\circ \quad \circ \quad \dots \quad \circ \quad a}^c \\ \bullet \quad \bullet \\ \hline \bullet \end{array} = \begin{array}{c} \overbrace{a \quad \dots \quad a}^b \quad \overbrace{a \quad \dots \quad a}^c \\ \bullet \quad \bullet \\ \hline \bullet \end{array} = \begin{array}{c} a(b+c) \\ \circ \\ \hline \bullet \end{array}$$

## 3.2 Multiplicative inverses

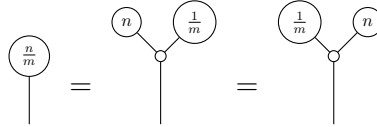
From Th. 3.1.3 it follows that the *tick* map behaves like a multiplicative inverse.

**Corollary 3.2.1.** 

Thus, it is possible to encode multiplicative inverses:

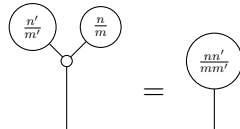


and providing that  $m \neq 0$ , positive fractions can be encoded as well:

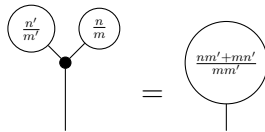


Using the basic properties of CFAs, the axioms of the GHZ/W-Calculus and the theorems given in the previous section, it can be easily shown that rational fractions behave like normal fractions with respect to addition and multiplicative laws. Detailed proofs can be found in [CKMR11] and very similar proofs can be found in the next chapter.

**Example 3.2.2.** *Multiplication*




**Example 3.2.3.** *Addition of fractions also behaves as expected:*



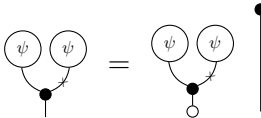
## 3.3 Additive inverses

Additive inverses can also be encoded in the GHZ/W-Calculus but require the addition of a new map,  $\dagger$ , which

- is involutive.
- is a phase for .

- is such that  $\circlearrowleft, \circlearrowright$  is a plugging set.

A map meeting all these conditions is suitable to encode additive inverses.

**Theorem 3.3.1.** 

Finally, the set of axioms required to encode rational arithmetic is complete once one defines

$$\begin{aligned} \begin{array}{c} \circlearrowleft \\ | \\ -\psi \end{array} &:= \begin{array}{c} \circlearrowright \\ | \\ \psi \end{array} \\ \begin{array}{c} \circlearrowleft \\ | \\ -\frac{n}{m} \end{array} &:= \begin{array}{c} \circlearrowright \\ | \\ \frac{n}{m} \end{array} \cdot \end{aligned}$$

**Remark** In  $\mathbf{FdHilb}$ ,  $\circlearrowright$  is  $-Z$ ,  $Z$  being the Pauli  $Z$  gate.

### 3.4 Conclusion

We gave here a quick summary of [CKMR11], the goal of this chapter being to introduce the important notions required to understand the next chapters. Let us recall here the simplified versions (no scalars) of Th.3.1.1, Th.3.1.2 and Th.3.1.3.

$$\begin{aligned} \begin{array}{c} \circlearrowleft \\ | \\ \psi \end{array} &= \begin{array}{c} \circlearrowright \\ | \\ \psi \end{array} \begin{array}{c} \circlearrowright \\ | \\ \psi \end{array} \\ \begin{array}{c} \circlearrowleft \\ | \\ \psi \end{array} &= \begin{array}{c} \circlearrowright \\ | \\ \frac{1}{\psi} \end{array} \end{aligned} \cdot$$

# Chapter 4

## Complex Arithmetic

We propose here an encoding of complex arithmetic in the GHZ/W Calculus as well as an alternative to the definition of additive inverses given in the previous chapter.

### 4.1 $\iota$

Suppose we have a point  $\uparrow : I \rightarrow Q$ . Suppose further that  $\uparrow$  verifies

$$\begin{array}{c} \blacktriangle \quad \blacktriangle \\ \curvearrowright \\ \bullet \\ | \end{array} = \bullet \quad (4.1)$$

up to a scalar. For a reason that will soon become clear we call this point  $\iota$ .

**Remark** All the equalities given in this chapter are given up to a scalar, providing they are invertible.

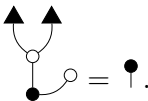
**Proposition 4.1.1.**  $\begin{array}{c} \blacktriangle \quad \blacktriangle \\ \curvearrowright \\ \bullet \\ | \end{array} = \bullet \Leftrightarrow \begin{array}{c} \blacktriangle \quad \blacktriangle \\ \curvearrowleft \\ \circ \\ | \end{array} = \bullet$

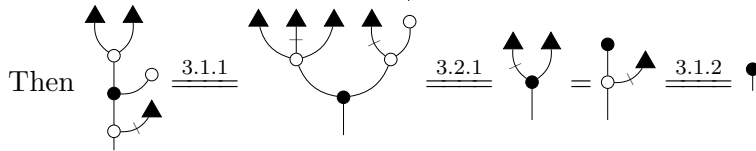
*Proof.* Proof by double implication.

Suppose that  $\begin{array}{c} \blacktriangle \quad \blacktriangle \\ \curvearrowright \\ \bullet \\ | \end{array} = \bullet$ .

Then  $\begin{array}{c} \blacktriangle \quad \blacktriangle \\ \curvearrowright \\ \circ \\ | \end{array} \xrightarrow{3.1.1} \begin{array}{c} \blacktriangle \quad \blacktriangle \quad \blacktriangle \\ \curvearrowright \quad \curvearrowright \\ \bullet \\ | \end{array} \xrightarrow{3.2.1} \begin{array}{c} \blacktriangle \quad \blacktriangle \\ \curvearrowleft \\ \bullet \\ | \end{array} = \begin{array}{c} \bullet \quad \blacktriangle \\ \curvearrowleft \\ \circ \\ | \end{array} \xrightarrow{3.1.2} \bullet$




Conversely, suppose that .

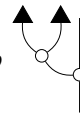


□

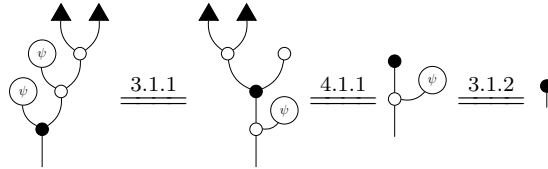
Intuitively, this proposition reveals the fact that, in normal complex arithmetic,  $i$  can be equivalently defined as being solution of  $i + \frac{1}{i} = 0$  or of  $i^2 + 1 = 0$ .

## 4.2 Additive Inverses

We propose a way to define additive inverses using .

**Proposition 4.2.1.** *The map  acts as an additive inverse operation.*

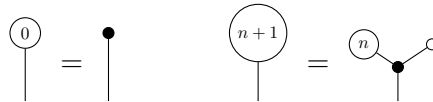
*Proof.* For any point  $\psi : I \rightarrow Q$



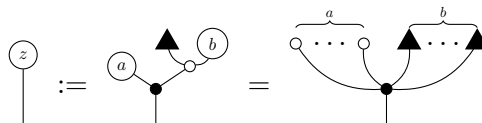
□


## 4.3 Encoding

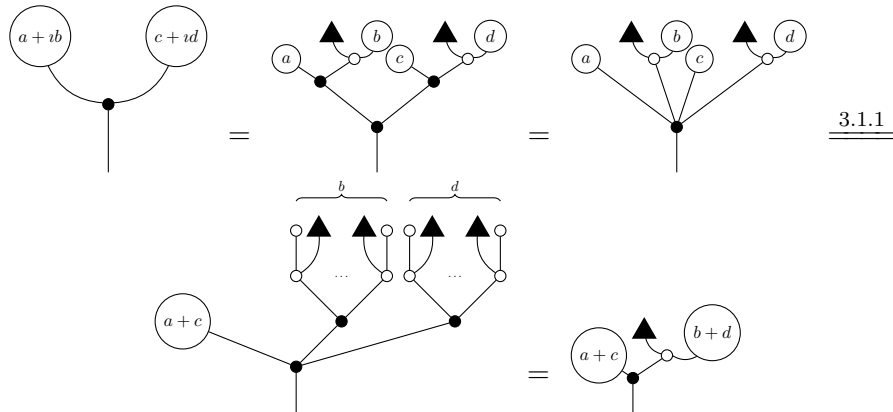
We propose here an encoding of complex numbers based on the encoding for natural numbers exposed in the previous chapter. Recall that positive numbers are encoded as follows:



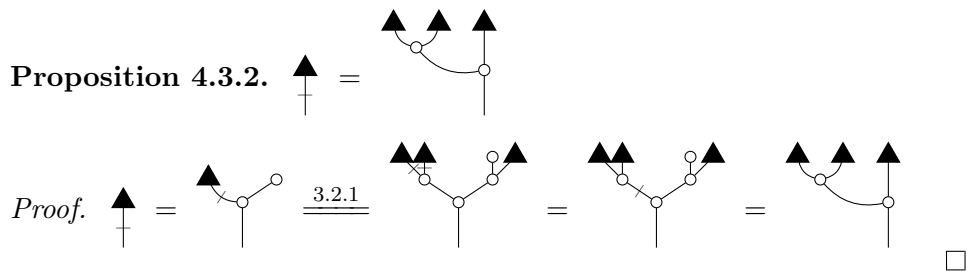
For  $z = a + ib$ ,  $(a, b) \in \mathbb{N}^2$  we define



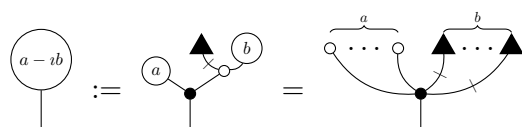
**Example 4.3.1.**  behaves like normal addition for complex numbers.

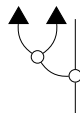


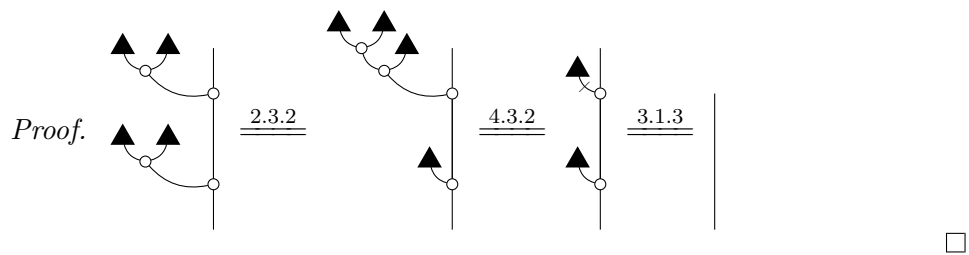
Using a very similar proof one can also show that  $(a + ib)(c + id) = ac - bd + i(ad + bc)$ .

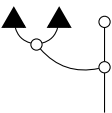



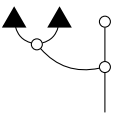

This proposition justifies a more compact encoding of negative imaginary numbers which does not use the additive inverse defined earlier and yet is consistent with the rest of the encoding.

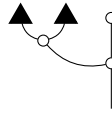



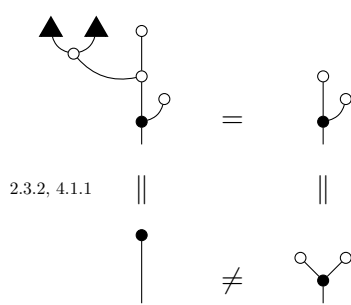
**Corollary 4.3.3.** The additive inverse map  is involutive.



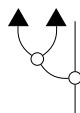

**Proposition 4.3.4.**  and  form a plugging set.

*Proof.* By contradiction. Suppose that  and  are proportional.

Then, since we write equalities up to a scalar,  =  and we have

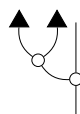

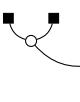


□

**Proposition 4.3.5.**  is a phase for .

*Proof.* Trivial using theorem 2.3.2.

□

We note that  verifies all the conditions to be an additive inverse map as defined in [CKMR11]. This shows that this encoding is consistent with the one defined in the previous chapter. Conversely, if a map  $\dagger$  is involutive, a phase for ,  $\{ \uparrow$  and  $\downarrow \}$  is a plugging set and the map is of the form  for some point  $\blacksquare : I \rightarrow Q$  then we have  $\blacksquare = \blacktriangle$ .

The proof is given in the annex.

**Remark**  $\iota$  being defined up to a scalar in 4.1, the illegible points  $|\psi\rangle = k(\alpha|0\rangle + \beta|1\rangle)$  ( $k, \alpha, \beta \in \mathbb{C}^* \times \mathbb{C}^2$  in **FdHilb**) are such that

$$\begin{cases} \alpha^2 + \beta^2 = 0 \\ \alpha\beta = 0. \end{cases}$$

The solutions are  $\{(\alpha = e^{i(\frac{\pi}{4} + \frac{n\pi}{2})}, \beta = \alpha e^{-i\frac{\pi}{2}}), n = 0, 1\} \cup \{(\alpha = e^{-i(\frac{\pi}{4} + \frac{n\pi}{2})}, \beta = \alpha e^{i\frac{\pi}{2}}), n = 0, 1\}$ . An interesting example of such a point being  $|\psi\rangle =$

$-i(e^{i\frac{3\pi}{4}}|0\rangle + e^{-i\frac{3\pi}{4}}|1\rangle)$  because the definition of additive inverses given in this encoding coincides with the one given in the previous chapter, *i.e.* minus the Z-Pauli gate.

## 4.4 Conclusion

We saw here that with the addition of only a point, the GHZ/W Calculus is powerful enough to approximate the complex field with arbitrary precision. The main advantage in this approach is that it enable ones to define imaginary numbers and additive inverses at the same time. Considering that the GHZ/W-calculus is based only on CFAs, this encoding is a nice demonstration of the expressiveness of this calculus.

# Chapter 5

## From Open Graphs to Quantomatic

As stated in the introduction, one of the many doors opened by the SMC formalism of quantum physics is the possibility to reason about graphs automatically through graph pattern matching and term rewriting. In order to give the reader an idea of what we mean by those terms we will start this chapter by showing how these techniques can be used on a concrete example.

### 5.1 An Intuitive Approach

Let us prove that  $\begin{array}{c} \bullet \\ | \\ \circ \\ | \\ \circ \\ | \\ \circ \end{array} = \begin{array}{c} \bullet \bullet \\ | \\ \vdash \end{array}$  [Roy11] and let us analyse carefully how reasoning about graphs works.

*Proof.* A detailed proof of this result is as follows:

$$\begin{array}{l}
 \begin{array}{c} \bullet \\ | \\ \circ \\ | \\ \circ \\ | \\ \circ \end{array} = \begin{array}{c} \bullet \\ | \\ \circ \\ | \\ \circ \\ | \\ \circ \\ | \\ \circ \end{array} \quad ( \begin{array}{c} \vdash \\ / \quad \backslash \end{array} = \begin{array}{c} \backslash \quad / \\ \circ \end{array} ) \\
 = \begin{array}{c} \bullet \\ | \\ \circ \\ | \\ \circ \\ | \\ \circ \end{array} \quad ( \vdash \text{ is involutive} ) \\
 = \begin{array}{c} \bullet \bullet \\ | \\ \circ \\ | \\ \circ \end{array} \quad ( \begin{array}{c} \bullet \\ | \\ \circ \\ / \quad \backslash \end{array} = \begin{array}{c} \bullet \bullet \\ | \\ \vdash \end{array} ) \\
 = \begin{array}{c} \bullet \bullet \\ | \\ \vdash \end{array}
 \end{array}$$

□

In each step a part of the graph is identified as being a match for one of the rules of the GZH/W-calculus. Then, the subgraph is replaced accordingly. In the axioms, the dangling wires (input/outputs) can be matched by anything. In the first rewrite, the input is plugged to  $\begin{array}{c} \bullet \\ | \\ \vdash \end{array}$  for instance. In

order to make this kind of process more rigorous and to automate it, a few important notions need to be defined:

- graphs are not usual graphs because edges need not to be connected at both ends ; they are *Open Graphs*
- *Pattern Matching* for Open Graphs
- *Graph Rewriting* of Open Graphs

All these definitions are necessary to understand what **quantomatic** [DD<sup>+</sup>] is, exactly, and how it works. That is why the rest of this chapter includes important results taken from the fields of pattern matching, graph theory and term rewriting. A thorough tutorial on these topics can be found in [DK10].

## 5.2 Open Graphs

Open Graphs extend the usual notion of graph as we know it. Again, we will choose a category theoretic approach to this topic. This approach yields equivalent, yet sometimes unusual, definitions.

**Definition** A *directed graph*  $G$  is a functor from the category with two objects and two arrows defined by  $\bullet \rightrightarrows \bullet$  to **Set**.

Consequently, the category of graphs is a functor category, and a morphism of graphs is a natural transformation. Unfolding this definition, the object on the left identifies the *Edges* of the graph and the one on the right identifies the *Vertices*. The arrows are used to match each edge with its corresponding *source* and *target*.

We recall here the definitions of *full* subcategories and *slice* categories. More information and examples about those can be found [HSB04] and [vO02].

**Definition** A category  $\mathcal{A}$  is a *full* subcategory of  $\mathcal{B}$  if for any  $A, A' \in \text{Ob}(\mathcal{A})$ ,  $\text{hom}_{\mathcal{A}}(A, A') = \text{hom}_{\mathcal{B}}(A, A')$ .

**Example 5.2.1.** *The category **Grp** of all groups and groups homomorphisms is a full subcategory of **Mon**, the category of all monoids and monoid morphisms. Indeed, **Grp** is a subcategory of **Mon**. Moreover, let  $f$  be a group homomorphism from  $(A, \cdot)$  to  $(A', *)$ . Then,  $f$  is such that  $f(x \cdot y) = f(x) * f(y)$  and  $f(x \cdot x^{-1}) = f(e) = f(x) * f(x)^{-1} = e'$ . Conversely, if  $f$  is a monoid morphism from the group  $(A, \cdot, e)$  to the group  $(A', *, e')$  then  $f(x \cdot y) = f(x) * f(y)$  and for any  $x \in A$ ,  $f(x^{-1} \cdot x) = f(e) = e' = f(x) * f(x^{-1})$ . The unicity of the inverse implies  $f(x^{-1}) = f(x)^{-1}$ .*

**Definition** Let  $A$  be an object in the category  $\mathcal{C}$ . The *slice* category  $(\mathcal{C}/A)$  is the category which has for objects all arrows  $g$  with codomain  $A$ . An arrow from  $g : B \rightarrow A$  to  $h : B' \rightarrow A$  is an arrow  $k : B \rightarrow B'$  such that

$$\begin{array}{ccc} B & \xrightarrow{k} & B' \\ & \searrow g & \swarrow h \\ & A & \end{array}$$

commutes.

Open Graphs are typed graphs: vertices in the usual sense are 'proper' vertices or *edge-points*. One can interpret edge-points as dummy points inserted along the edges. Thus, in its most simple definition, an open-graph admits only two types of vertices. As suggested in [DK10], we call  $\mathcal{G}_2$  the following object of **Graph**:

$$\mathcal{G}_2 := \left( \textcircled{v} \right) \begin{array}{c} \curvearrowright \\ \curvearrowleft \end{array} \left( \textcircled{\epsilon} \right) \begin{array}{c} \curvearrowright \\ \curvearrowleft \end{array} .$$

**Definition** The category **OGraph** is the full subcategory of the slice category **Graph**/ $\mathcal{G}_2$ .

Open graphs are the objects of **OGraph**.

**Proposition 5.2.2.** [DK10] **OGraph** has coproducts, given by disjoint union of the underlying typed graphs.

**Definition** The set *inputs* (resp *outputs*) of an open graph  $G$ , noted  $In(G)$  (resp  $Out(G)$ ) is the set of in-edges (resp out-edges) of  $G$ . An *in-edge* (resp *out-edge*)  $e$  is an edge-point such that there does not exist any edge which admits  $e$  as target (resp source).

**Definition** The *boundary graph* of an open graph is the graph  $B := In(G) + Out(G)$ . By definition, the coproduct induces a unique map  $b : B \rightarrow G$ .  $b$  is called the *boundary map* of  $G$ .

**Definition** Two open-graphs  $L$  and  $R$  share the same boundary graph,  $B$ , by boundary maps  $b_1$  and  $b_2$ , when  $L \xleftarrow{b_1} B \xrightarrow{b_2} R$ ,  $In(L) \cong In(R)$ ,  $Out(L) \cong Out(R)$  and

$$\begin{array}{ccccc} & In(L) & \xrightarrow{\sim} & In(R) & \\ & \textcircled{0} & & \textcircled{1} & \\ & \swarrow l^i & & \searrow r^i & \\ L & \xleftarrow{l} & B & \xrightarrow{r} & R \\ & \swarrow l^o & & \searrow r^o & \\ & \textcircled{3} & & \textcircled{4} & \\ & Out(L) & \xrightarrow{\sim} & Out(R) & \\ & \textcircled{5} & & & \end{array}$$

commutes. Note that the commutativity of triangles 0, 2, 3 and 5 is given by the coproduct structure of  $B$ .

The list of definitions given above is quite dense; that is why we illustrate them here and show that they define beautifully the intuitive notions given in the first section.

$$L := \begin{array}{c} \bullet \rightarrow \circ \rightarrow \bullet \\ \uparrow \quad \uparrow \\ l_w \quad l_x \\ \uparrow \quad \uparrow \\ l_w^i \quad l_x^o \\ \uparrow \quad \uparrow \\ l_y \quad l_z \\ \uparrow \quad \uparrow \\ l_y^i \quad l_z^o \end{array}, R := \begin{array}{c} \bullet \quad \bullet \\ \uparrow \quad \uparrow \\ r_w \quad r_y \\ \uparrow \quad \uparrow \\ r_w^i \quad r_y^o \\ \uparrow \quad \uparrow \\ r_z \quad r_z \\ \uparrow \quad \uparrow \\ r_z^i \quad r_z^o \end{array}$$

$L$  and  $R$  share the same boundary: The boundary of  $L$  (resp  $R$ ) is given by the point-graph  $l_w, l_x, l_y, l_z$  (resp  $r_w, r_y, r_z$ ). The boundary maps of  $L$  and  $R$  are represented under the graphs using the dashed arrows. Obviously,  $In(L) \cong In(R)$  and  $Out(L) \cong Out(R)$ . Take, for instance the relations  $\rho_1 = (l_w^i, r_w^i), (l_y^i, r_y^i)$  and  $\rho_2 = (l_x^o, r_w^o), (l_z^o, r_z^o)$ . Having done that, the commutativity of triangles 1 and 4 becomes trivial. Recall that commutativity of triangles 0, 2, 3 and 5 is given by construction,  $B$  being a coproduct.

### 5.3 Matching

As seen earlier in the introductory section of this chapter, the notion of *matching* allows to identify a graph to a part of a larger graph. We give here the definition of a matching in the context of open graphs as it is defined in [DK10].

**Definition** The *edge neighbourhood*  $N(v)$  of a vertex  $v$  is the set of all edges that are connected to  $v$  (incoming or outgoing).

**Remark** Recall that what we call a vertex in the context of open graphs is a point of type  $V$ . Edge-points are not included here.

**Definition** Let  $G, H \in \mathcal{Ob}(\mathbf{OGraph})$  and  $f \in Hom_{\mathbf{OGraph}}(G, H)$ .  $f$  is a *local isomorphism* when, for every vertex  $v$  (*i.e.* point of type  $V$ ), the restriction of the edge function of  $f$  to  $N(v)$  is a bijection :  $f^v : N(v) \xrightarrow{\sim} N(f(v))$ .

**Definition** Let  $G, H \in \mathcal{Ob}(\mathbf{OGraph})$ . A monomorphism  $m : G \rightarrow H$  is called a *matching* when it is a local isomorphism. We say that  $G$  matches  $H$  at  $m$ .

**Definition** A span  $H_1 \xleftarrow{f} G \xrightarrow{g} H_2$  is called *boundary coherent* when  $f$  and  $g$  are matchings and

- $\forall p \in In(G)$  at least one of  $f(p)$  and  $g(p)$  is an input



- $\forall p \in \text{Out}(G)$  at least one of  $f(p)$  and  $g(p)$  is an output

All the tools necessary to perform operations on open graphs are now defined. Given a pair of open graphs two operations are possible: merging and subtraction. All these operations are part of the rewriting process of graphs in the GHZ/W Calculus and will be illustrated at the end of this chapter. The following definitions are taken from [DK10], where they are given with more details and followed by related theorems.

**Definition** Let  $H_1 \xleftarrow{m_1} G \xrightarrow{g} H_2$  be a boundary coherent span. The pushout

$$\begin{array}{ccc} M & \xleftarrow{m'_2} & H_2 \\ m'_1 \uparrow & & \uparrow m_2 \\ H_1 & \xleftarrow{m_1} & G \end{array}$$

is called merging of  $H_1$  and  $H_2$  on  $G$  by  $m_1$  and  $m_2$  and is noted  $M := G_1 +_{m_1, m_2} G_2$ .

If  $G$  is a point-graph then the merging is called a *plugging* and is noted  $M := H_1 +_{m_1, m_2}^* G$ .

**Remark** Recall that apart from the category theoretic definition of graphs given above, a graph  $G$  can be equivalently defined a 4--tuple  $(P, E, s, t)$ ,  $P$  and  $E$  being the sets of points and edges respectively and  $s$  and  $t$  being the maps which associate each edge with its source and target respectively.

**Definition** The subtraction of  $G$  from  $M$  at a match  $m : G \rightarrow M$ , written  $M -_m G$  is the open graph  $H$  defined as follows:

$$\begin{aligned} P_H &= (P_M \setminus m[P_G]) + P_B \\ E_H &= (E_M \setminus m[E_G]) \\ s_H(e) &= \begin{cases} b^o(p) & \text{if } p \in \text{Out}(G) \text{ and } m(p) = s_M(e) \\ s_M(e) & \text{otherwise} \end{cases} \\ t_H(e) &= \begin{cases} b^i(p) & \text{if } p \in \text{In}(G) \text{ and } m(p) = t_M(e) \\ t_M(e) & \text{otherwise} \end{cases} \end{aligned}$$

## 5.4 Rewriting

**Definition** A span  $L \xleftarrow{b_1} G \xrightarrow{b_2} R$  such that  $L$  and  $R$  share the same boundary is called a *rewrite rule*. The rule is said to *rewrite*  $G$  to  $G'$  at matching  $m$  when  $G'$  is defined by the double pushout as follows:

$$\begin{array}{ccccc}
L & \xleftarrow{b_1} & B & \xrightarrow{b_2} & R \\
\downarrow m & & \downarrow & & \downarrow \\
G & \xleftarrow{\quad} & G & \xrightarrow{-m} & L & \longrightarrow & G'
\end{array}$$

This expresses the fact that  $L$  is removed from  $G$  and replaced by  $R$  to form a new graph  $G'$ .

We now have the required framework to automatically apply rewrite rules to the open graphs used in the GHZ/W-calculus. However, in concrete use cases, the need to express infinite sets of rewrite rules emerges. This is required in order to express the spider rule, for instance : LHS must match a vertex with any number of inputs. !-boxes aim at solving this issue [DD09]: any part of the graph which is inside a !-box can be replicated any number of times, all the vertices inside the !-box being connected to the outside in the same way. Rewrite rules expressed using !-boxes allow graph pattern matching: a single rule can match more than one graph.

The set of graphs generated by a !-box graph is given by any sequence of the following operations:

- Copy: copy a !-box and the incident edges
- Drop: unbang vertices
- Merge: merge two !-box
- Kill: remove a !-box and its content

`Quantomatic` implements the Copy, Drop and Kill operations and allows to reason about !-graphs. Figure 5.1 shows how the spider rule for vertices of type  $GHZ$  is implemented in `quantomatic`. The graph on the left is the LHS and the graph on the right, the RHS. Thus, the rule is implemented by induction and the merging of multiple requires many applications.

## 5.5 Quantomatic

`Quantomatic` is a tool which automates the graph rewriting process : given a set of rewrite rules and a graph, it solves the sub-graph isomorphism problem in order to find which rules' LHSs are matchings for the graph and rewrites the graph accordingly. Technical details of the implementation of `quantomatic` and its capabilities are given in the next chapter. Here we will see, through an example, how `quantomatic` implements the techniques given above.

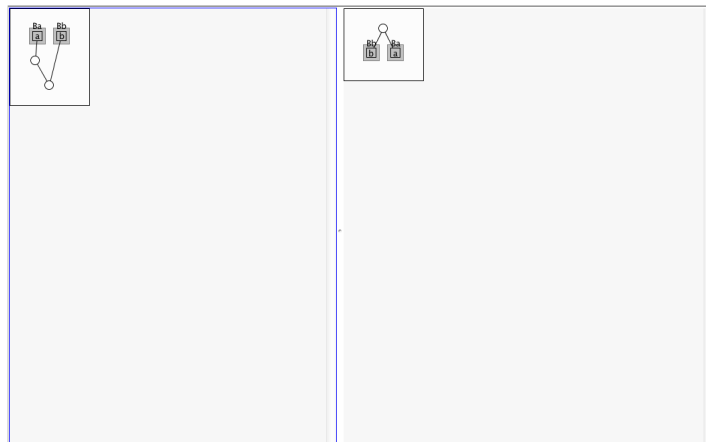


Figure 5.1: Rewrite rule making use of !-boxes in order to express the spider rule for GHZ vertices. Banged vertices are called *boundary* vertices.

**Remark** In this chapter we studied open graphs typed by  $\mathcal{G}_2$ . Those graphs admit only one type of vertex. It is possible to define and reason about graphs typed by more complex type-graphs [DK10]. Consider, for example the graphs used in the GHZ/W-calculus : vertices can be of type GHZ or W.

Let us consider the rewriting of



by the rule

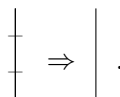
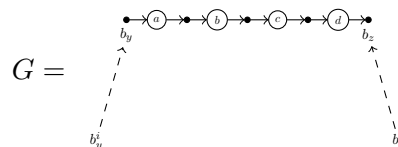


Figure 5.2 shows what these graphs look like in *quantomatic*. The vertices labelled  $c$  and  $d$  are the boundary vertices. The identity map is then represented by two boundary vertices connected by a wire. Seen as an open graph and introducing the "dummy" edge-points, here is the graph that our example is based upon:



In this graph, the vertices  $a$  and  $d$  are of type *GHZ* while the vertices  $b$  and  $c$  are of type *TICK*.

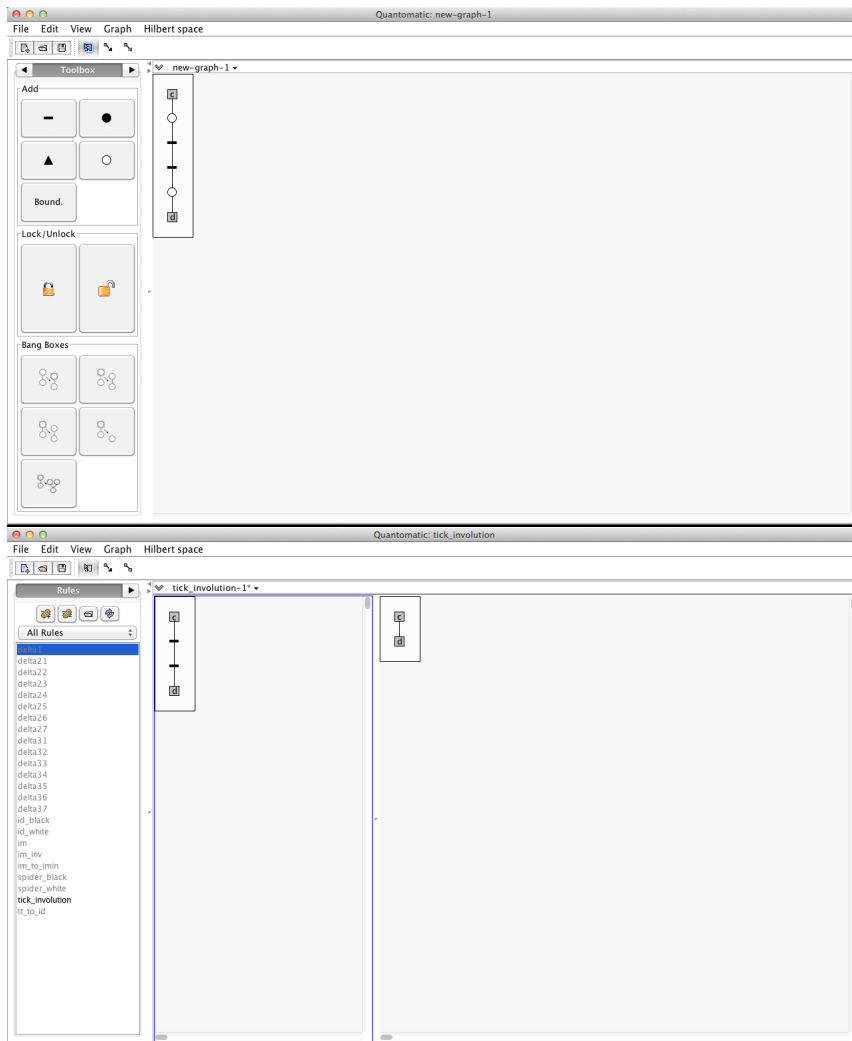
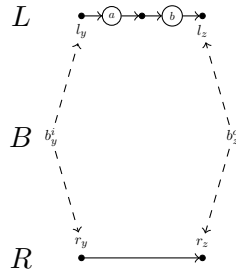


Figure 5.2: Top: Graph being rewritten. Bottom: Rewrite rule implemented in quantomatic.

### 5.5.1 Rewrite Rule

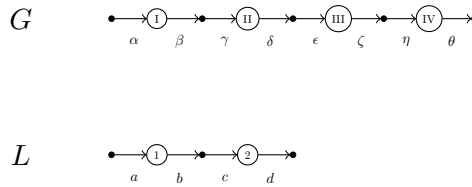
We consider the following span, the boundary maps being defined by the dashed lines :



The fact that  $In(L) \cong In(R)$  and  $Out(L) \cong Out(R)$  is implied by the graph. Thus the span  $L \leftarrow G \rightarrow R$  is a rewrite rule.

### 5.5.2 Matching

The next step is to show that there exists a matching  $L \rightarrow G$ . Let us use the following names on  $G$  and  $L$ :

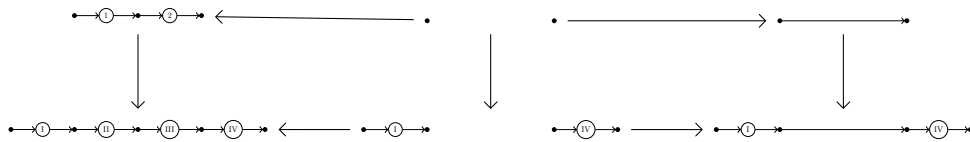


We define  $f : L \rightarrow G$ , such that  $f(1) = \text{II}$   $f(2) = \text{III}$  (we are interested only in vertices). Then  $L$  matches  $G$  at  $f$ .

Indeed, on the one hand we have  $N(1) = \{a, b\}$  and  $N(2) = \{c, d\}$  and on the other hand  $N(\text{II}) = \{\gamma, \delta\}$  and  $N(\text{III}) = \{\epsilon, \zeta\}$ . Thus it is clear that  $f^1(N(1)) \xrightarrow{\sim} N(f^1(1))$  and  $f^2(N(2)) \xrightarrow{\sim} N(f^2(2))$ .

### 5.5.3 Rewriting

Finally, knowing that  $G$  can be rewritten by  $L \Rightarrow R$  at  $f$ , we can rewrite it by double pushout:



**Remark** This example requires to extend slightly the definitions given above as  $G$  has four vertices with two different types.

### 5.5.4 Automation

quantomatic recreates this process:

- **Share the same boundray :** Checked when the set of rules is loaded
- **Matching :** The software finds all possible matchings and applies the first match by alphabetical order. See Figure 5.3
- **Rewriting :** The graph is rewritten by successive application of rewrite rules. See Figure 5.4,

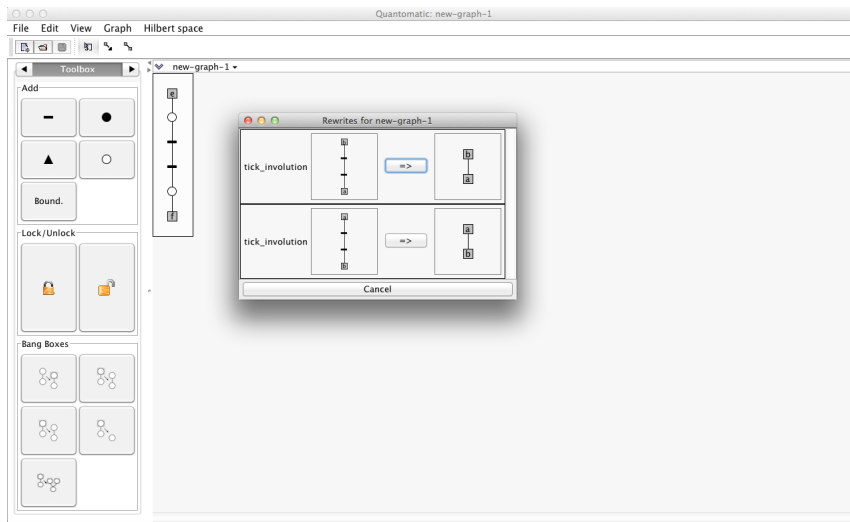


Figure 5.3: Quantomatic proposes a list of matching rules. The user can choose to apply a specific rule or just apply automatically the first one.

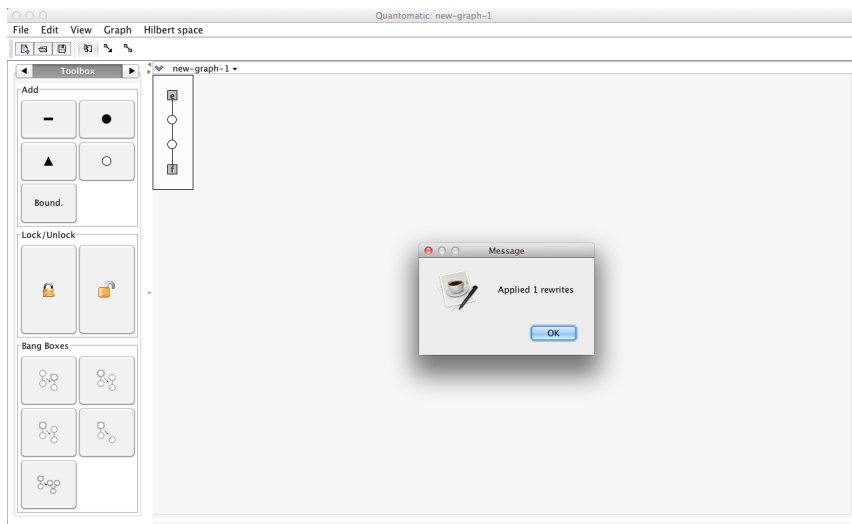


Figure 5.4: The rule is applied and the graph rewritten.

## Chapter 6

# Implementation in Quantomatic

So far we have mainly described the theoretical aspects of `Quantomatic`. This chapter aims at giving a technical account of the software explaining which modifications were required in order to automate the theories presented in chapters 3 and 4.

### 6.1 Coding

#### 6.1.1 Description of the software

`Quantomatic` is divided in two parts:

##### Overview

- The front-end that we usually call `GUI`.
- The back-end that we usually call `Core`.

The core is written in Poly/ML and can be run as a standalone application through a console interface. The rewrite rules and the graphs are exported/imported using an XML format. A typical session would be:

1. Launch the Core.
2. Load a set of rules, called *ruleset*.
3. Create a new graph or load an existing one.
4. Normalize.

Thanks to the GUI, written in JAVA, it is possible to visualize graphs and communicate more easily with the core : it provides a user friendly interface with the core.



## Graph Param

We will not describe in detail the implementation of **Quantomatic**. There is, however, a structure which is of great interest to us : **GRAPH\_PARAM**. This is the structure which determines which theory the software will reason about. Whenever a new theory is declared, one needs to create a new instance of **GRAPH\_PARAM** and specify, among other things, what the types of the vertices and of the edges are. Each vertex type is identified by a string used for human interactions.

For instance, in the case of the GHZ/W-calculus, there are two types of vertices, GHZ (identified by 'GHZ') and W (identified by 'W'), and they do not carry any data. The edges are also of type 'unit'.

## Graphs

As mentioned above, graphs are stored in an XML format. They can be loaded and saved from both the core and the gui. In the end, the core is the one actually in charge of generating/parsing the XML code.

The structure of a graph file is very informative as it mirrors the structures defined by the core.

```
<graph>
  <vertex name=" name_of_the_vertex">
    <type>GHZ</type>
    <data/>
  </vertex>
  <vertex name=" another_name">
    <type>edge-point</type>
    <data/>
  </vertex>
  ...
  <edge name=" name_of_the_edge" dir=" true"
        source=" name_of_source_vertex"
        target=" name_of_target_vertex">
    <type>unit</type>
    <data/>
  </edge>
  ...
  <bangbox name=" name_of_the_bangbox">
    <vertex>name_of_banged_vertex_1</vertex>
    <vertex>name_of_banged_vertex_2</vertex>
    ...
  </bangbox>
```

</graph>

Note the possibility to specify whether an edge is directed. Of course, the types of the vertices must correspond to the types declared by the theory loaded by the core.

In figure 6.1 we show how a graph is represented by the GUI. In this case, the graph is taken from the X/Z-calculus in which vertices carry data. This explains the coloured labels represented under the nodes.

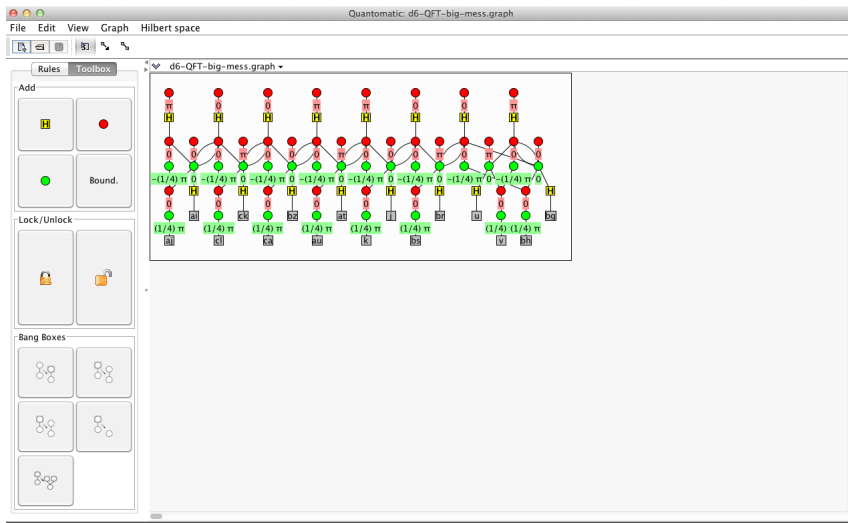


Figure 6.1: A graph displayed by the GUI.

## Rulesets

Rulesets are also defined in XML files. Within the file, each rule must have a unique name and it is possible to 'tag' them in order to have groups of rules which are conceptually close. Some rules can be enabled by default.

The structure of a ruleset is the following:

```
<ruleset>
  <allrules>
    <rule>
      <name>Name_of_rule_1</name>
      <definition>
        <lhs>Here the LHS...</lhs>
        </rhs>Here the RHS...</rhs>
      </definition>
    </rule>
  </allrules>
</ruleset>
```

```

</rule>
...
<tags>Rules can be tagged here</tags>
<active>
    <rule>
        Name_of_rule_active_by_default_1
    </rule>
    <rule>
        Name_of_rule_active_by_default_2
    </rule>
    ...
</active>
</allrules>
</ruleset>

```

Figure 6.2 shows how rulesets are displayed. A list of rules is available in the side-bar and rules can be enabled/disabled by tag or by selection. It is also possible to open a specific rule.



Figure 6.2: A ruleset represented in the GUI.

### 6.1.2 Support for multiple theories

When **Quantomatic** was designed its goal was to implement the  $X/Z$ -calculus [CD11], also called the Red-Green Calculus. Thus, this theory was hardcoded in both the core and the gui. A substantial amount of work has been done in order to support multiple theories. As suggested above, the ruleset and the **GRAPH\_PARAM** must agree on the theory which is in use. When it comes to represent the graphs, the GUI must also support different representations for the vertices. It is now possible to implement a new theory like

the GHZ/W-calculus in a matter of minutes, thanks to the work that has been done over the past few months. The hardcoded part in the core has been reduced to only a few lines and the GUI supports a change of theory on-the-fly.

The author worked mainly on the GUI and implemented what we will call *theory visualization*. Indeed, each graphical calculus supported by `quantomatic` is characterized by the shape and the color of its vertices. In order to be able to switch easily from an arbitrary theory to another, a theory visualization is defined in an XML file which specifies all the data required by both the core and the GUI. Thus, implementing a new theory is actually defining a consistent triple (**GRAPH\_PARAM**, Ruleset, Theory Visualization).

The structure of a theory-visualization file is the following:

```
<theory name="name_of_this_visualization"
      implements="
        name_of_the_implemented_GRAPH_PARAM">
  <nodetype name="name_of_the_vertex_type">
    <data type="None_or_String_or_
      MathExpression"></data>
    <visualization>
      <node svgFile="node.svg"/>
      <label fill="ffffff_or_any_
        color"/>
    </visualization>
  </nodetype>
  ...
</theory>
```

In Figure 6.3 we see that the toolbox is dynamically created when the theory-visualization file is loaded. It provides an easy access to !-boxes operations.

### 6.1.3 Perspectives

The implementation of a 2.0 version of `Quantomatic` is under active development and will support theory switching on-the-fly from both the core and the GUI.

As new theories are implemented new needs are emerging. For instance, the "plugging" technique presented earlier is not something that can be automated. In order to use this technique efficiently one would need to implement a kind of proof strategy.

The construction of graphs representing rational or complex expressions in the GHZ/W-calculus is time consuming and error prone. However, they obey simple rules and could be automated. Similarly, if someone works frequently with quantum logic gates, it could be handy to build graphs

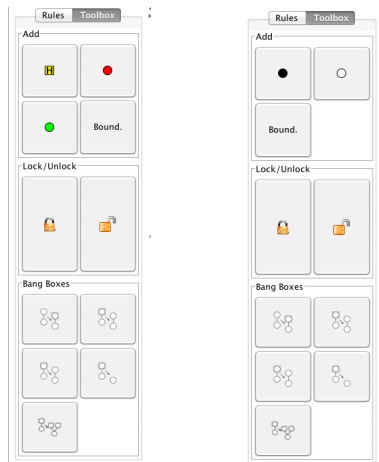


Figure 6.3: The toolbox with two different theory visualizations. Left: X/Z. Right: GHZ/W

automatically corresponding to boolean expressions. Plug-ins, which one could see as macros, may help to solve those issues. A proof-of-concept has been developed and has been used in order to automatically build graphs corresponding to rational or complex expressions.

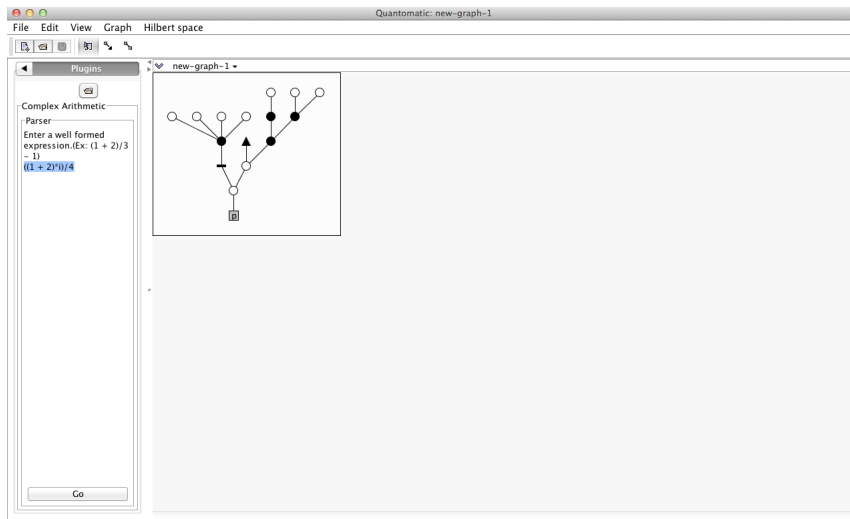


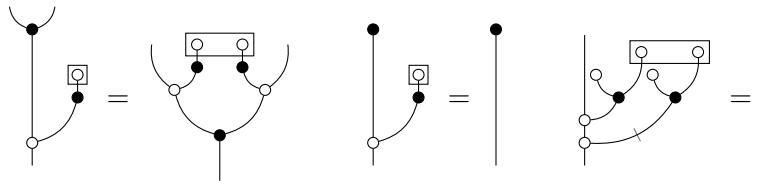
Figure 6.4: Proof-of-concept for the plugin system. The complex expression entered on the left is parsed using a JavaScript script and the graph is generated on the right.

## 6.2 Rewrite Rules for complex and rational arithmetic

The theories presented in Chapters 3 and 4 admit rewrite rules expressed in term of graph patterns described in Chapter 5.

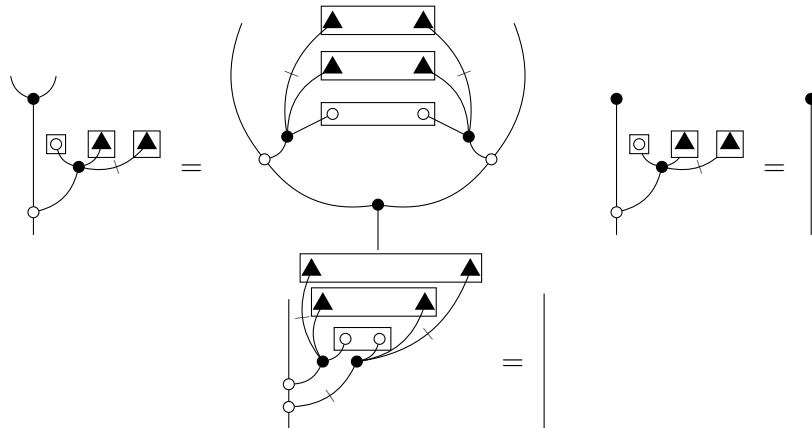
### 6.2.1 Rational arithmetic

The following rewrite rules were given in [CKMR11]:

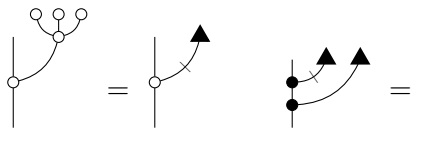


### 6.2.2 Complex Arithmetic

We give here our modified version of the rules which describe rational arithmetic in the GHZ/W-Calculus.



Note that the case  $z/0$  is not avoided by this reduced set of rules. For that to happen it would be required to split the second rule into three different rules. Moreover, the more compact definition of  $\iota$  given above is expressed by:



### 6.3 Implementation and Results

Even if succeeding in implementing arbitrary theories represents an important step towards a full implementation of rational arithmetic, one last point should be clarified : in the first rule given above, provided the graph is an undirected one, LHS matches RHS and the software enters an infinite loop. For this reason it is important to use directed graphs in `Quantomatic` (which was not the case so far) and to implement the directed versions of the spider theorem and of all the rules mentioned in the previous section.

Finally, a version implementing rational arithmetic and set up to work out of the box is available at

`https://benjaminfrot@github.com/benjaminfrot/quantomatic.git`  
, branch `arithdemo`:

```
> git clone https://benjaminfrot@github.com/benjaminfrot/quantomatic.git
> cd quantomatic
> git checkout arithdemo
```

Figure 6.5 (resp 6.6) shows how complex (resp rational) arithmetic is implemented in `quantomatic` and how a typical rewriting works: the user creates a graph and then normalises it. It is possible to stop the process at any moment and `quantomatic` will display the number of rewrites that were applied.

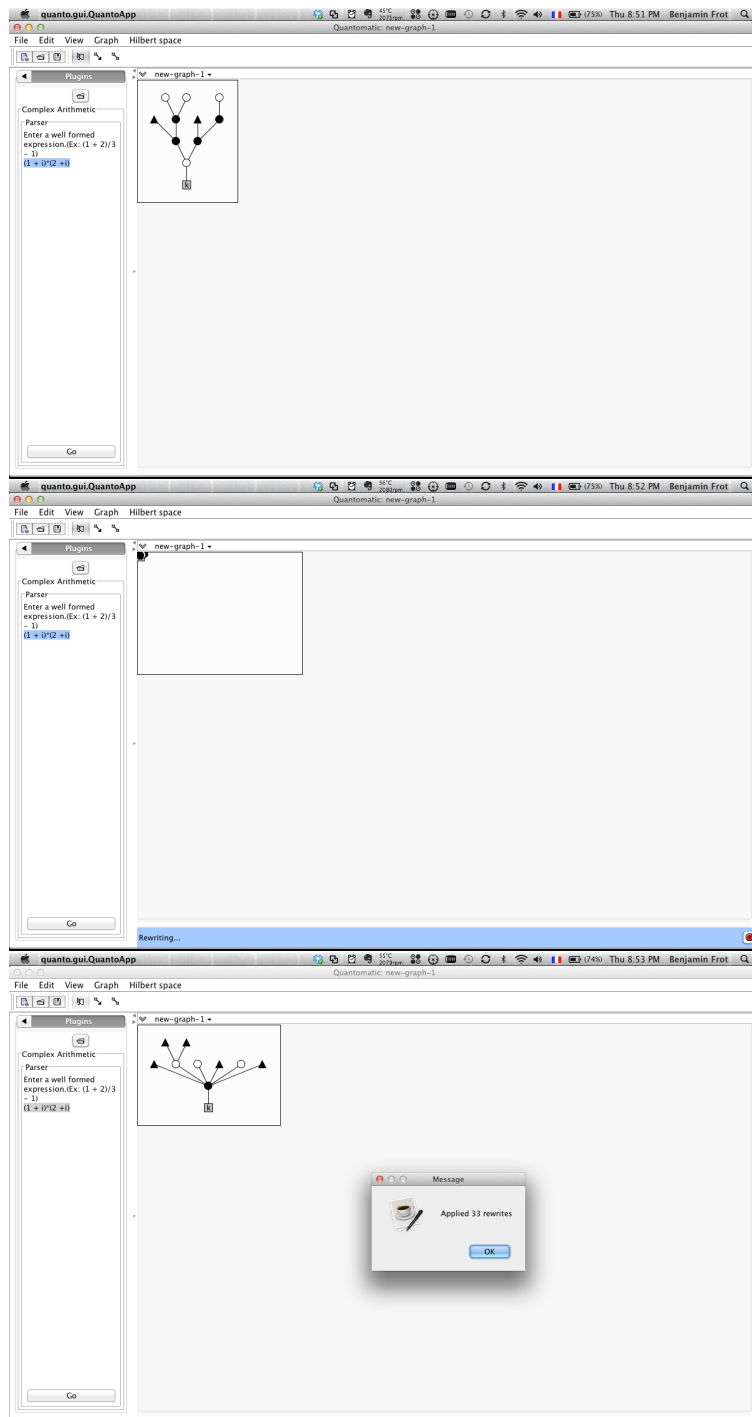


Figure 6.5: Complex Arithmetic: Top: Graph created by the complex arithmetic plug-in and corresponding to  $(1 + i)(2 + i)$ . Middle: Graph being rewritten. Bottom: Result,  $3i + 2 + i^2$ .



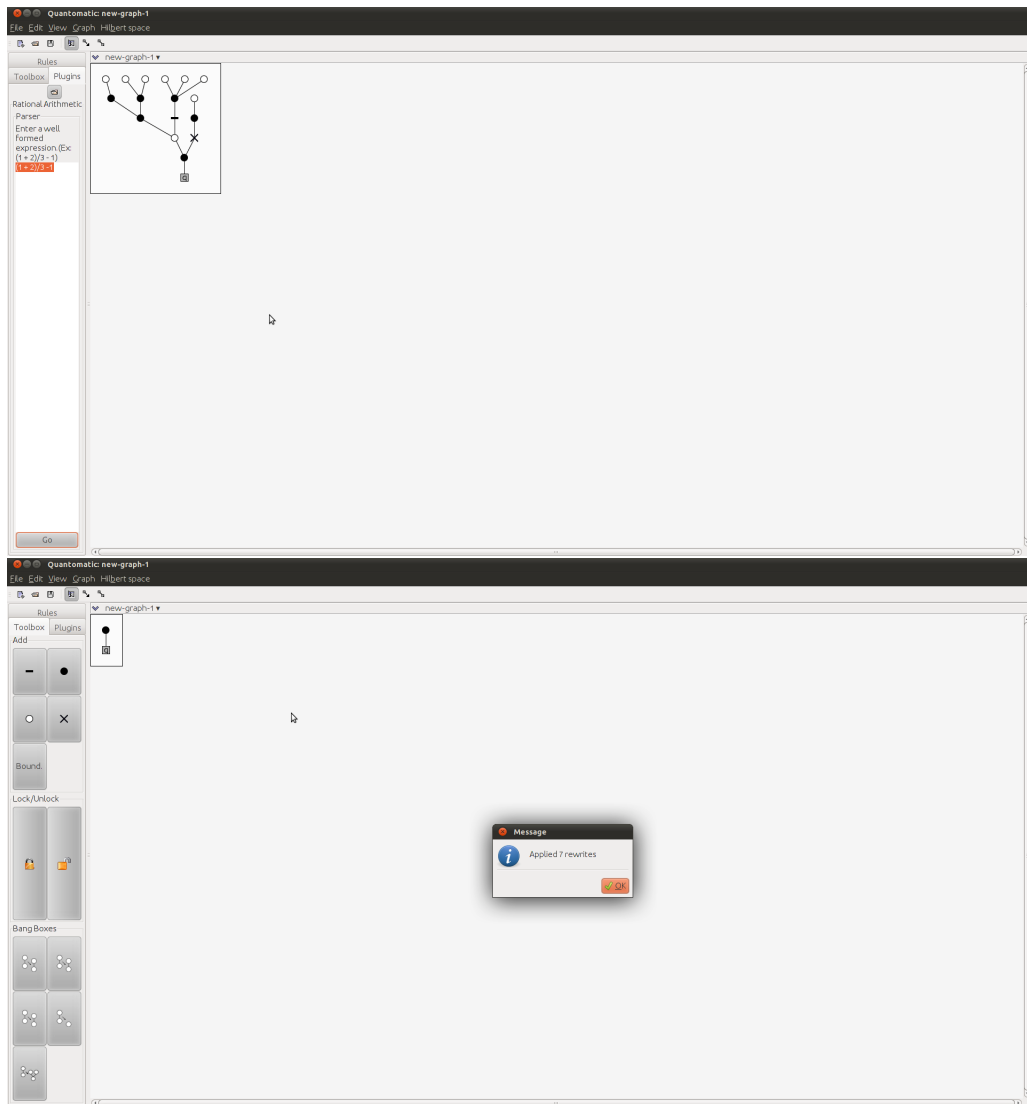


Figure 6.6: Rational Arithmetic: Top: Graph created by the rational arithmetic plug-in and corresponding to  $(1 + 2)/3 - 1$ . Bottom: Result, 0.

## Chapter 7

# Conclusion

We discussed the possibility to encode rational arithmetic within the GHZ/W-calculus and proposed an extension to that encoding. We then gave an account of the mathematical foundations of **Quantomatic**. Finally, we showed how the encodings exposed in the first half of this dissertation were implemented in **Quantomatic** and saw that this represents a significant milestone for the project.

In the introduction, we mentioned two approaches to reading this work:

- The reader who is familiar with the notions discussed earlier will probably notice that the GHZ/W-calculus extended with a minor extension is able to approximate the real or complex fields with an arbitrary precision. This gives an idea of the expressiveness and power of that language, which is based only on the CFAs, when compared to the X/Z-calculus, for instance.

Another important result of this dissertation is the fact **Quantomatic** has reached a significant milestone: an easy and fast implementation of arbitrary theories. Thus, this software is becoming both a theorem prover for Quantum Mechanics and tool for the researcher who wants to test the consistency of a new theory.

- The less experienced reader will probably remember how *Categorical Quantum Mechanics* and the tools it allows to define (graphical calculus, automated reasoning) can be used in order to describe quantum systems. Again, this formalism should be compared to the Hilbert space formalism initiated by von Neumann.

Future work will follow two directions : the theoretic aspect and **Quantomatic**. Indeed, even if the implementation of complex arithmetic is a nice demonstration of what can be done within the GHZ/W-calculus, it would be interesting to see whether these encodings can be used to prove new results. **Quantomatic** will be released in its version 2.0 very soon. However, new features such as plug-ins and strategies

would be useful in improving the user-interface and the power of the tool respectively.

# Bibliography

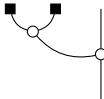
- [AC04] Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, pages 415–425, Washington, DC, USA, 2004. IEEE Computer Society.
- [AC05] Samson Abramsky and Bob Coecke. Abstract physical traces. *THEORY AND APPLICATIONS OF CATEGORIES*, 14(6):111–124, 2005.
- [AT11] Samson Abramsky and Nikos Tzevelekos. Introduction to Categories and Categorical Logic. February 2011.
- [BBC<sup>+</sup>93] Charles H. Bennett, Gilles Brassard, Claude Crpeau, Richard Jozsa, Asher Peres, and William K. Wootters. Teleporting an unknown quantum state via dual classical and epr channels, 1993.
- [Ben92] Charles H. Bennett. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Physical Review Letters*, 69(20):2881, 1992.
- [CD11] Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.
- [CE11a] Bob Coecke and Bill Edwards. Three qubit entanglement within graphical Z/X-calculus. *Electronic Proceedings in Theoretical Computer Science*, 52:22–33, March 2011.
- [CE11b] Bob Coecke and Bill Edwards. Toy quantum categories (extended abstract). *Electronic Notes in Theoretical Computer Science*, 270(1):29 – 40, 2011. Proceedings of the Joint 5th International Workshop on Quantum Physics and Logic and 4th Workshop on Developments in Computational Models (QPL/DCM 2008).

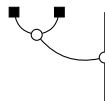
- [CK10] Bob Coecke and Aleks Kissinger. The compositional structure of multipartite quantum entanglement. *Lecture Notes in Computer Science*, 6199:31, 2010.
- [CKMR11] Bob Coecke, Aleks Kissinger, Alex Merry, and Shibdas Roy. The GHZ/W-calculus contains rational arithmetic. *Electronic Proceedings in Theoretical Computer Science*, 52(Hpc 2010):34–48, March 2011.
- [Coe05] Bob Coecke. Kindergarten quantum mechanics, 2005. arXiv:quant-ph/0510032v1.
- [Coe09] Bob Coecke. Quantum pictorialism. *Contemporary Physics*, 51:59–83, 2009. arXiv:0908.1787.
- [CP09] Bob Coecke and Eric Oliver Paquette. Categories for the practising physicist. *Arxiv preprint arXiv09053010*, quant-ph(0905.3010v2):105, 2009.
- [CW87] A. Carboni and R.F.C. Walters. Cartesian bicategories i. *Journal of Pure and Applied Algebra*, 49(1-2):11 – 32, 1987.
- [DD<sup>+</sup>] Lucas Dixon, Ross Duncan, , Matvey Soloviev, Benjamin Frot, Alex Merry, and Aleks Kissinger. Quantomatic : <http://sites.google.com/site/quantomatic/>.
- [DD09] Lucas Dixon and Ross Duncan. Graphical reasoning in compact closed categories for quantum computation. *Annals of Mathematics and Artificial Intelligence*, 56(1):23–42, July 2009.
- [DK10] Lucas Dixon and Aleks Kissinger. Open Graphs and Monoidal Theories. *arXiv:1011.4114v1*, 2010.
- [Doe10] Andreas Doering. Quantum computer science, 2010. <http://www.cs.ox.ac.uk/teaching/courses/2010-2011/quantum/>.
- [DVC00] W. Dür, G. Vidal, and J. I. Cirac. Three qubits can be entangled in two inequivalent ways. *Physical Review A*, 62(6):12, November 2000.
- [Her10] Michael Herrmann. Models of Multipartite Entanglement. *MSc Thesis, University of Oxford*, (September), 2010. <http://www.cs.ox.ac.uk/people/bob.coecke/Michael.pdf>.
- [HSB04] Horst Herrlich, George E. Strecker, and Bernhard Banaschewski. Abstract and concrete categories. the joy of cats, 2004.

- [JS91] Andre Joyal and Ross Street. The geometry of tensor calculus, i. *Advances in Mathematics*, 88(1):55–112, 1991.
- [KL80] Max Kelly and M. L. Laplaza. Coherence for Compact Closed Categories. *Journal of Pure and Applied Algebra*, 19:193–213, 1980.
- [Koc03] Joachim Kock. *Frobenius algebras and 2D topological quantum field theories*, volume 59 of *London Mathematical Society student texts*. Cambridge University Press, 2003.
- [Lan98] Saunders M. Lane. *Categories for the Working Mathematician (Graduate Texts in Mathematics)*. Springer, 2nd edition, September 1998.
- [Pen71] Roger Penrose. Applications of negative dimensional tensors. *Combinatorial Mathematics and its Applications*, 1971.
- [Roy11] Shibdas Roy. Towards normal forms for ghz/w calculus. *Computing Research Repository*, abs/1104.2, 2011.
- [Sel07] Peter Selinger. Dagger compact closed categories and completely positive maps. *Electron. Notes Theor. Comput. Sci.*, 170:139–163, March 2007.
- [Sel09] Peter Selinger. A survey of graphical languages for monoidal categories. August 2009.
- [Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26:1484–1509, October 1997.
- [Spe07] Robert W. Spekkens. In defense of the epistemic view of quantum states: a toy theory. *PHYS.REV.A*, 75:032110, 2007.
- [vN96] John von Neumann. *Mathematical Foundations of Quantum Mechanics*. Princeton University Press, translation - from german edition, October 1996.
- [vO02] Jaap van Oosten. Basic category theory, 2002. <http://www.staff.science.uu.nl/ooste110/syllabi/catsmoeder.pdf>.

# Appendix A

## Proof for 4.3

We give here the proof for the claim made in 4.3: If a map  $\dagger$  is involutive, a phase for  $\circlearrowleft$ ,  $\circlearrowright$  and  $\circlearrowright$  is a plugging set and the map is of the form  for some point  $\dagger : I \rightarrow Q$  then we have  $\dagger = \uparrow$ .

*Proof.* Let us suppose that the map  meets all the conditions stated above. Then, according to [CKMR11], it acts as an additive inverse and for  $\circlearrowright$  we have

$$\begin{aligned} \Rightarrow \quad & \begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} = \begin{array}{c} \bullet \\ | \end{array} \\ & \begin{array}{c} \text{Diagram 3} \\ \text{Diagram 4} \end{array} = \begin{array}{c} \bullet \\ | \end{array} \end{aligned}$$

□

## Appendix B

# Code Listing

The following script can be used to easily verify whether two maps, expressed either in graphical notation or Dirac notation, are equal.

```
#!/usr/bin/python

from numpy import kron, matrix
from math import sqrt
import sys

#####

def otimes(a, b):
    return kron(a, b)

#####

def main():

    #A few useful definitions
    zero = matrix("1_ ; _0")
    one = matrix("0_ ; _1")
    zeroT = zero.transpose()
    oneT = one.transpose()
    plus = (zero + one)/sqrt(2)
    minus = (zero - one)/sqrt(2)
    plusT = plus.transpose()
    minusT = minus.transpose()

    ##### GHZ/W Calculus
    #####

    #deltaGHZ |00><0| + |11><1|
    deltaGHZ = kron(zero, zero)*zeroT + kron(one, one)*
    oneT
    #epsilonGHZ <0| + <1|
```



```

epsilonGHZ = zeroT + oneT
#muGHZ |0><00| + |1><11|
muGHZ = zero*kron(zeroT, zeroT) + one*kron(oneT,
    oneT)
#etaGHZ |0> + |1>
etaGHZ = zero + one

#deltaW |00><0| + |01><1| + |10><1|
deltaW = kron(zero, zero)*zeroT + kron(zero, one)*
    oneT + kron(one, zero)*oneT
#epsilonW <0|
epsilonW = zeroT
#muW |1><11| + |0><01| + |0><10|
muW = one*kron(oneT, oneT) + zero*kron(zeroT, oneT)
    + zero*kron(oneT, zeroT)
#etaW |1>
etaW = one

#tick
tick = matrix("0_1_;\_1_0")
#swap
swap = matrix("1_0_0_0_;\_0_0_1_0_;\_0_1_0_0_;\_0_0_0_
    1")
#id
id = matrix("1_0_;\_0_1")

#Extra definitions
capGHZ = deltaGHZ * etaGHZ
cupGHZ = epsilonGHZ * muGHZ
capW = deltaW * etaW
cupW = epsilonW * muW

X = matrix("0_1_;\_1_0")

if(len(sys.argv) < 2):
    print "script \_ 'expr1' \_ ['expr2']"
    print 'Ex: \_python\_g2m.py \_ "deltaGHZ \_ * \_ etaW"
    ,
    print 'or: \_python\_g2m.py \_ "deltaGHZ \_ * \_ etaW" \_
    "otimes(etaW, \_ etaW)" '
    print "\_ * \_ = \_ \_ circle \_ ; \_ otimes(x, \_ y) = x \_
    otimes \_ y"
    print ""
    print " Available: \_ deltaGHZ, \_ muGHZ, \_
    epsilonGHZ, \_ etaGHZ, \_ deltaW, \_ muW, \_
    epsilonW, \_ etaW, \_ tick, \_ swap, \_ id, \_ capGHZ, \_
    cupGHZ, \_ capW, \_ cupW"
    print ""
    sys.exit(0)

```

```

leftHandside = eval(sys.argv[1])

print sys.argv[1] + "\n" + str(leftHandside)
print ""

if (len(sys.argv) > 2):
    rightHandside = eval(sys.argv[2])

    print sys.argv[2] + "\n" + str(
        rightHandside)
    print ""
    try:
        print (leftHandside ==
                rightHandside).flatten().all()
    except:
        print False

if __name__ == "__main__":
    main()

```